ORACLE
PeopleSoft

PeopleTools 8.58: Component Interfaces

May 2020

ORACLE

PeopleTools 8.58: Component Interfaces

Contents

Preface: Preface ..... **vii**
Understanding the PeopleSoft Online Help and PeopleBooks vii
Hosted PeopleSoft Online Help vii
Locally Installed Help vii
Downloadable PeopleBook PDF Files vii
Common Help Documentation vii
Field and Control Definitions viii
Typographical Conventions viii
ISO Country and Currency Codes viii
Region and Industry Identifiers ix
Translations and Embedded Help ix
Using and Managing the PeopleSoft Online Help x
Understanding PeopleSoft Component Interfaces x
PeopleTools Related Links x
Contact Us x
Follow Us xi
Chapter 1: Getting Started with PeopleSoft Component Interfaces **13**
Overview 13
Implementing PeopleSoft Component Interfaces 13
Implementing the Excel to Component Interfaces Utility 14
Chapter 2: Understanding Component Interfaces **15**
Understanding Component Interfaces 15
Component Interface Architecture 15
Component Interface Attributes 15
Name 16
Keys 16
Properties 16
Collections 16
Methods 16
Component Interface Definitions and Views 17
Chapter 3: Developing Component Interfaces **21**
Understanding Stateful CI 21
Creating Component Interface Definitions 21
Understanding Creating Component Interface Definitions 21
Creating New Component Interfaces 22
Naming Component Interface Definitions 24
Associating Component Interfaces with Menus 24
Determining the Fields to Expose in Component Interfaces 25
Using Keys 26
Understanding Keys 26
Adding and Deleting Keys 27
Setting Properties 28
Understanding Standard Properties 28
Creating User-Defined Properties 37
Deleting User-Defined Properties 37
Renaming User-Defined Properties 38

Creating Reference Properties....39
Making Properties Read-Only....41
Working with Collections....41
Working with Methods....42
Understanding Session Functions and Methods....42
Understanding Standard Methods....43
Understanding Collection Methods....46
Enabling and Disabling Standard Methods....49
Creating User-Defined Methods....50
Exporting User-Defined Methods....52
Validating Component Interfaces....52
Setting Security Options....53
Testing Component Interfaces....55
Searching Component Interfaces to Test....55
Testing Component Interfaces....58
Determining ItemByKeys Parameters....60
Understanding Synchronization....61
Writing Component Interface Programs....62
Understanding Runtime Considerations....63
General Considerations....63
Scope Conflicts....63
Interactive Mode....64
Chapter 4: Programming Component Interfaces Using PeopleCode....65
Understanding PeopleCode Behavior and Limitations....65
PeopleCode Event and Function Behavior....65
CopyRowset Language Considerations....66
Limitations of Client-Only PeopleCode....67
Generating PeopleCode Templates to Access Component Interfaces....67
Understanding PeopleCode Templates....68
Chapter 5: Programming Component Interfaces in Java....71
Building APIs in Java....71
Setting Up the Java Environment....71
Generating Java Runtime Code Templates....72
Understanding the Java Template....74
Chapter 6: Programming Component Interfaces in C++....77
Building APIs for C++....77
Setting Up the C++ Environment....77
Setting Up Client Machines to Access C++ APIs....78
Configuring Compilers for C++ Projects....78
Generating C++ Runtime Code Templates....79
Understanding the C++ Template....80
Chapter 7: Using the Component Interface Software Development Kit....85
Understanding the Component Interface SDK....85
Component Interface SDK Samples....85
Prerequisites for Using the Component Interface SDK....85
Using the SDK_BUS_EXPENSES Test Page....86
Testing the SDK_BUS_EXP Component Interface....86
Using the Component Interface SDK Sample in Java and C+ +....87
Understanding using the Component Interface SDK Samples in Java and C++....87
Building the Component Interface SDK Sample (Java)....87
Building the Component Interface Sample (C++)....87

Running the Component Interface SDK Sample in Java and C++....87
Interpreting the Code for the Component Interface SDK Sample (Java)....88
Interpreting the Code for the Component Interface SDK Sample (C++)....90
Chapter 8: Using the Excel-to-Component Interface Utility....93
Understanding the Excel-to-Component Interface Utility....93
Prerequisites for Using the Excel to CI Utility....93
Understanding Building Component Interfaces for the Excel to Component Interface Utility....94
Testing Component Interfaces....94
Performance Expectations....95
PeopleCode Behavior and Limitations....95
Default Properties....95
Running the Excel to Component Interface Utility....96
Granting Access to the WEBLIB_SOAPTOCI iScript....96
Enabling the Developer Menu in Microsoft Excel 2007 and Later Versions....96
Enabling Macros in Microsoft Excel....97
Starting the Excel to Component Interface Utility....98
Converting Excel to Component Interface Utility Templates to the Current Excel Version....98
Viewing the Excel to Component Interface Coversheet....98
Setting Up Connection Information....99
Entering Connection Information....99
Translations and Multilingual Support....102
Connecting to the Database to Create a Template and Submit Data....103
Creating Templates....103
Understanding the Template Actions Toolbar....105
Entering Data into the Template....107
Entering Data on the Data Input Sheet....108
Using the Data Input Sheet....108
Viewing the Staged Data....109
Correcting and Resubmitting Data....110
Creating SOAP/XML Requests....111
Request Format....111
Sample Create Request....111
Sample Get Request....112
Sample Update Request....112
Sample Updatedata Request....112
Sending Requests....113
Receiving Responses....113
Viewing a Response if a Row Already Exists....113
Viewing a Sample Get Request and Response....114
Diagnosing and Resolving Errors....114
Viewing Log Files....114
Viewing HTML Page....115
Resolving Error Messages for Client Environments....115
Appendix A: Creating Component Interface-Based Services....117
Understanding Generating Component Interfaced-Based Services....117
Appendix B: Using Services to Validate Prompt Table and Translate Field Values....119
Understanding Validating Prompt Table and Translate Field Values....119
Prerequisites for Validating Prompt Table and Translate Field Values....120
Validating Prompt Table Field Values....120
Understanding Validating Set Control Fields....120
Using the PTLOOKUPPROMPT Service Operation....121

Using the PTLOOKUPPROMPT_REST_GET Service Operation....121
Validating Translate (XLAT) Field Values....122
Understanding Translate (XLAT) Table Entries....123
Understanding Security When Validating Translate (XLAT) Field Values....123
Using the PTLOOKUPXLAT Service Operation....123
Using the PTLOOKUPXLAT_REST_GET Service Operation....124
Using Messages to Request Valid Prompt Field and Translate (XLAT) Field Values....125
Using Response Messages to Retrieve Valid Prompt Field and Translate (XLAT) Field Values....126
Examples: Validating Prompt Field and Translate (XLAT) Field Values....127
Example 1: Validating a Translate (XLAT) Field....127
Example 2: Performing a Prompt Table Lookup with a Field Value Wildcard....128
Example 3: Filtering Field Values by Name/Value Pairs....129
Example 4: Specifying Set Control Field Values to Validate Field Values Controlled by Set Control Fields....130
Example 5: Specifying Set Control ID Values to Validate Field Values Controlled by Set ID Values....131

Preface

Understanding the PeopleSoft Online Help and PeopleBooks

The PeopleSoft Online Help is a website that enables you to view all help content for PeopleSoft applications and PeopleTools. The help provides standard navigation and full-text searching, as well as context-sensitive online help for PeopleSoft users.

Hosted PeopleSoft Online Help

You can access the hosted PeopleSoft Online Help on the Oracle Help Center. The hosted PeopleSoft Online Help is updated on a regular schedule, ensuring that you have access to the most current documentation. This reduces the need to view separate documentation posts for application maintenance on My Oracle Support. The hosted PeopleSoft Online Help is available in English only.

To configure the context-sensitive help for your PeopleSoft applications to use the Oracle Help Center, see Configuring Context-Sensitive Help Using the Hosted Online Help Website.

Locally Installed Help

If you're setting up an on-premise PeopleSoft environment, and your organization has firewall restrictions that prevent you from using the hosted PeopleSoft Online Help, you can install the online help locally. See Configuring Context-Sensitive Help Using a Locally Installed Online Help Website.

Downloadable PeopleBook PDF Files

You can access downloadable PDF versions of the help content in the traditional PeopleBook format on the Oracle Help Center. The content in the PeopleBook PDFs is the same as the content in the PeopleSoft Online Help, but it has a different structure and it does not include the interactive navigation features that are available in the online help.

Common Help Documentation

Common help documentation contains information that applies to multiple applications. The two main types of common help are:

- Application Fundamentals
- Using PeopleSoft Applications

Most product families provide a set of application fundamentals help topics that discuss essential information about the setup and design of your system. This information applies to many or all applications in the PeopleSoft product family. Whether you are implementing a single application, some combination of applications within the product family, or the entire product family, you should be familiar with the contents of the appropriate application fundamentals help. They provide the starting points for fundamental implementation tasks.

In addition, the *PeopleTools: Applications User's Guide* introduces you to the various elements of the PeopleSoft Pure Internet Architecture. It also explains how to use the navigational hierarchy, components, and pages to perform basic functions as you navigate through the system. While your application or implementation may differ, the topics in this user's guide provide general information about using PeopleSoft applications.

Field and Control Definitions

PeopleSoft documentation includes definitions for most fields and controls that appear on application pages. These definitions describe how to use a field or control, where populated values come from, the effects of selecting certain values, and so on. If a field or control is not defined, then it either requires no additional explanation or is documented in a common elements section earlier in the documentation. For example, the Date field rarely requires additional explanation and may not be defined in the documentation for some pages.

Typographical Conventions

The following table describes the typographical conventions that are used in the online help.

Typographical Convention	***Description***
Key+Key	Indicates a key combination action. For example, a plus sign (+) between keys means that you must hold down the first key while you press the second key. For Alt+W, hold down the Alt key while you press the W key.
. . . (ellipses)	Indicate that the preceding item or series can be repeated any number of times in PeopleCode syntax.
{ } (curly braces)	Indicate a choice between two options in PeopleCode syntax. Options are separated by a pipe (|).
[ ] (square brackets)	Indicate optional items in PeopleCode syntax.
& (ampersand)	When placed before a parameter in PeopleCode syntax, an ampersand indicates that the parameter is an already instantiated object. Ampersands also precede all PeopleCode variables.
⇒	This continuation character has been inserted at the end of a line of code that has been wrapped at the page margin. The code should be viewed or entered as a single, continuous line of code without the continuation character.

ISO Country and Currency Codes

PeopleSoft Online Help topics use International Organization for Standardization (ISO) country and currency codes to identify country-specific information and monetary amounts.

ISO country codes may appear as country identifiers, and ISO currency codes may appear as currency identifiers in your PeopleSoft documentation. Reference to an ISO country code in your documentation

does not imply that your application includes every ISO country code. The following example is a country-specific heading: "(FRA) Hiring an Employee."

The PeopleSoft Currency Code table (CURRENCY_CD_TBL) contains sample currency code data. The Currency Code table is based on ISO Standard 4217, "Codes for the representation of currencies," and also relies on ISO country codes in the Country table (COUNTRY_TBL). The navigation to the pages where you maintain currency code and country information depends on which PeopleSoft applications you are using. To access the pages for maintaining the Currency Code and Country tables, consult the online help for your applications for more information.

Region and Industry Identifiers

Information that applies only to a specific region or industry is preceded by a standard identifier in parentheses. This identifier typically appears at the beginning of a section heading, but it may also appear at the beginning of a note or other text.

Example of a region-specific heading: "(Latin America) Setting Up Depreciation"

Region Identifiers

Regions are identified by the region name. The following region identifiers may appear in the PeopleSoft Online Help:

- Asia Pacific
- Europe
- Latin America
- North America

Industry Identifiers

Industries are identified by the industry name or by an abbreviation for that industry. The following industry identifiers may appear in the PeopleSoft Online Help:

- USF (U.S. Federal)
- E&G (Education and Government)

Translations and Embedded Help

PeopleSoft 9.2 software applications include translated embedded help. With the 9.2 release, PeopleSoft aligns with the other Oracle applications by focusing our translation efforts on embedded help. We are not planning to translate our traditional online help and PeopleBooks documentation. Instead we offer very direct translated help at crucial spots within our application through our embedded help widgets. Additionally, we have a one-to-one mapping of application and help translations, meaning that the software and embedded help translation footprint is identical—something we were never able to accomplish in the past.

Using and Managing the PeopleSoft Online Help

Select About This Help in the left navigation panel on any page in the PeopleSoft Online Help to see information on the following topics:

- Using the PeopleSoft Online Help
- Managing Hosted online help
- Managing locally installed PeopleSoft Online Help

Understanding PeopleSoft Component Interfaces

A PeopleSoft component interface is a PeopleTools definition that you create in PeopleSoft Application Designer. It enables synchronous access to a PeopleSoft component from another application.

PeopleTools Related Links

PeopleTools 8.58 Home Page

PeopleTools Elasticsearch Home Page

"PeopleTools Product/Feature PeopleBook Index" (PeopleTools 8.58: Getting Started with PeopleTools)

PeopleSoft Hosted Online Help

PeopleSoft Information Portal

PeopleSoft Spotlight Series

PeopleSoft Training and Certification | Oracle University

My Oracle Support

Oracle Help Center

Contact Us

Send your suggestions to psoft-infodev_us@oracle.com. Please include the applications update image or PeopleTools release that you're using.

Follow Us

Facebook.

YouTube

Twitter@PeopleSoft_Info.

PeopleSoft Blogs

LinkedIn

Chapter 1

Getting Started with PeopleSoft Component Interfaces

Overview

A component interface is a set of application programming interfaces (APIs) that you can use to access and modify PeopleSoft database information programmatically. PeopleSoft component interfaces expose a PeopleSoft component (a set of pages grouped for a business purpose) for synchronous access from another application (PeopleCode, Java, or C/C++). A PeopleCode program or an external program (Java, or C/C++) can view, enter, manipulate, and access PeopleSoft component data, business logic, and functionality. Additionally, you can use the Component Interface Tester to check the validity of your component interface and the Excel to Component Interface Utility to manage your data.

Component interfaces are created in PeopleSoft Application Designer, so you should ensure that you are familiar with PeopleTools and Application Designer.

This section provides information to consider before you begin to use PeopleSoft component interfaces. In addition to implementation considerations presented in this section, take advantage of all PeopleSoft sources of information, including the installation guides, release notes, and PeopleBooks.

Related Links

"PeopleSoft Application Designer Overview" (PeopleTools 8.58: Application Designer Developer's Guide)

"Using PeopleSoft Application Designer" (PeopleTools 8.58: Application Designer Developer's Guide)

Implementing PeopleSoft Component Interfaces

PeopleSoft PeopleTools include the functionality to create component interfaces for your applications.

Complete the following tasks before you begin creating component interfaces for your implementation:

- Install your Application according to the installation guide for your database type.

  See the PeopleSoft installation guide for your platform and product line.

- Establish a user profile that gives you access to PeopleSoft Application Designer and any other processes that you will use. See "Working With User Profiles" (PeopleTools 8.58: Security Administration).

Implementing the Excel to Component Interfaces Utility

PeopleSoft provides the Excel to Component Interface utility that enables you to upload data from Microsoft Excel into your PeopleSoft database. Several tasks are involved in setting up the Excel to Component Interfaces Utility.

See Running the Excel to Component Interface Utility.

Chapter 2

Understanding Component Interfaces

Understanding Component Interfaces

A component interface enables exposure of a PeopleSoft component (a set of pages grouped together for a business purpose) for synchronous access from another application (such as PeopleCode, Java, C/C++, or XML). Component interfaces can be viewed as "black boxes" that encapsulate PeopleSoft data and business processes, and hide the details of the underlying page and data. Component interfaces can be used to integrate one application with another application or with external systems. Component interfaces execute the business logic built into the component and as a result, they provide a higher level of data validation than a simple SQL insert.

A component interface maps to one, and only one, PeopleSoft component. However, you can create multiple component interfaces for the same component. You create component interfaces in PeopleSoft Application Designer. Record fields on the component are mapped to the keys and properties of the component interface. Methods are used to find, create, modify, or delete data.

Component Interface Architecture

The component interface architecture comprises three fundamental elements—components, component interfaces, and the component interface API.

Every component interface has the following main attributes:

- Name.
- Keys (Get keys, Create keys, and Find keys).
- Properties and collections (fields and records).
- Methods.

Note: In most cases, component interfaces act like their associated components, meaning that PeopleCode events typically trigger in the same order as the component. However, several runtime exceptions relate to component interfaces and PeopleCode processing and search dialog box processing.

Related Links

"Understanding the Component Processor" (PeopleTools 8.58: PeopleCode Developer’s Guide)

Component Interface Attributes

This section discusses the name, keys, properties, collections, and methods of component interfaces.

Name

Each component interface requires a unique name that is specified when the component interface is created. The calling programs use the name of the component interface to access properties and methods.

Keys

Keys are special properties containing values that retrieve an instance (Get keys) or a list of instances (Find keys) of the component interface. When you create a new component interface, Get and Find keys are created based on the search record definition for the underlying component. However, you can add, remove, or change keys in PeopleSoft Application Designer. Create keys are created for components that have the Add action enabled.

Properties

Properties provide access to both component data and component interface settings. Component interfaces include two types of properties: standard and user-defined.

- Standard properties are assigned automatically when a component interface is created.

  Standard properties can be set to true or false. These properties are not displayed in the PeopleSoft Application Designer. Examples of standard properties include InteractiveMode, GetHistoryItems, and EditHistoryItems.

- User-defined properties map to record fields on the PeopleSoft component and are displayed in the PeopleSoft Application Designer.

  A property can correspond to a field or a scroll (collection). You can control which user-defined properties are included on the component interface.

Note: Every PeopleSoft Application Designer definition—including the component interface—has a definition properties dialog box in which you make design-time settings for the definition. Those properties should not be confused with the runtime properties that are discussed here.

Collections

A component interface collection is a special type of property that corresponds to a scroll. It contains fields and subordinate scrolls as defined in its underlying component. By default, each collection uses the name of the primary record for the underlying scroll.

Methods

A method is a function that performs a specific task on a component interface at runtime. As with component interface properties, two main types of methods are available: standard and user-defined. For example, you can use methods to save or create a new purchase order. Runtime access to each method is determined by the security that you have for that specific method.

- Standard methods are those that are available for all component interfaces.

  The Find, Get, Save, and Cancel methods are automatically generated by PeopleSoft Application Designer when a new component interface is created. The Create method is created for components

that have the Add action enabled. In the component interface designer, standard methods are highlighted in gray.

- User-defined methods are created in PeopleSoft Application Designer to provide added functionality to the component interface.

  These methods are functions that are made accessible through the component interface. Each function maps to a user-defined method. In the component interface designer, user-defined methods are highlighted in blue.

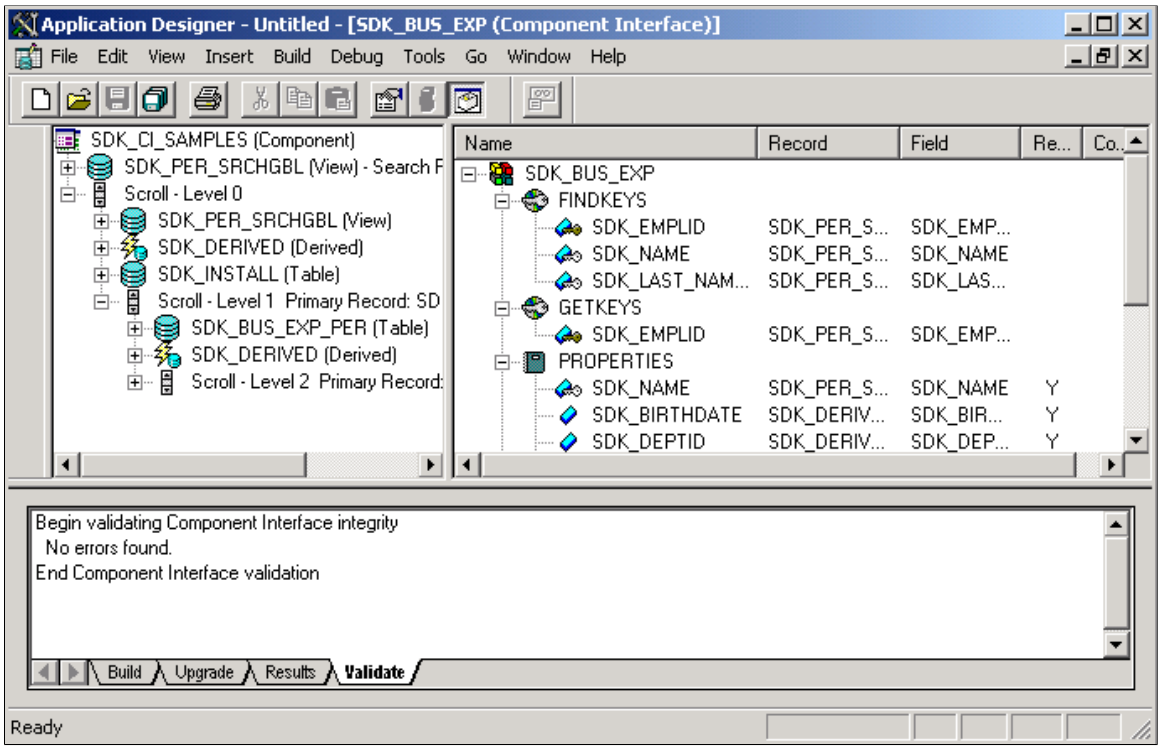
Component Interface Definitions and Views

You create, modify, and review your component interface definition by using PeopleSoft Application Designer. You open the component interface definition just as you would any other definition, such as a page or record.

When working with a component interface definition in PeopleSoft Application Designer, you see the component view on the left and the component interface view on the right.

Image: PeopleSoft Application Designer

This example shows the component and component interface view in PeopleSoft Application Designer.



The component view shows records and scrolls in the component, using a tree representation. The structure is the same as the one you see on the structure tab of a component in PeopleSoft Application Designer. Drag the fields and collections that you want exposed to the component interface view.

The component interface view shows the exposed keys, properties, and methods, using a tree representation. When you open a component interface, properties are displayed in the order in which they appear in the component view.

The tree nodes in both the component view and the component interface view have different icons. Some icons are used in both the component view and the component interface view with slightly different meanings. The following tables explain the meaning of each icon and column in the component interface view.

Component Interface View Icons

This table lists the component interface view icons:

	Component interface.
	Group of keys.
	Property that is a key field from the underlying record.
	Alternate search key.
	Group of properties or methods.
	Collection.
	Property or user-defined method.
	Standard method.
	Property that is a required field for the underlying record.
	Item in a component interface that is no longer in sync with the underlying component. For example, if a field on which a property depends is deleted from the component, this icon appears.

Component Interface View Columns

The following terms describe the columns in the component interface view.

Name	Name of a specific element of a component interface (such as the name of a property or method). The default name for field properties is the field name. The default name for collections is the primary record name.
Record	Name of the underlying record on which a specific element is based. If the underlying record name changes, the component interface continues to point to the appropriate record.
Field	Name of the field to which a component interface property points. Like the record name, the underlying field name can

	change, and the component interface continues to point to the appropriate field.
Read Only	*Y* in this column indicates that a specific property has been marked read-only.
Comment	Identifies comments that exist in the Edit Property dialog box for the selected key, property, or collection.

Note: In the component interface view, properties appear in the same order as they appear in the component and are not sorted alphabetically.

Chapter 3

Developing Component Interfaces

Understanding Stateful CI

For any PeopleTools release before PeopleTools 8.55, the Component interfaces could be defined as local variables. When you initialized the local variable it required to load the legacy component every time. For which you faced delays and the system performance suffered because when the component interface with large and complex legacy component was called from multiple events, the local variable was initialized multiple times in each event.

If you wanted to change the data in the legacy component, then you had to call the **Save** method of the component interface each time.

Considering the complexity involved in accessing the component interface object from multiple events, it is feasible to load the component interface one time. For which, you should declare the component interface as a Stateful CI which means declare the component interface as a component or global variable. Benefits of using a Stateful CI are:

- Easy to reuse in different events.
- One- time loading of the legacy component.
- Data changes saved across different events and different server trips.

The performance of the component interface improves because:

- There is no need to instantiate the component interface instance in every local scope.
- There is no need to load data for legacy component used in the CI.
- There is no need to run PreBuild and PostBuild events number of times.
- There is no need to call **Save** method in every local scope to keep the data changes.

Creating Component Interface Definitions

This section discusses how to create and work with component interface definitions.

Understanding Creating Component Interface Definitions

This section discusses key concepts for creating component interface definitions.

Component Structure

You must know the structure of the component for which you are constructing a component interface because each component interface refers to a single component. You can use an existing component within an application or create a new one for the sole purpose of constructing a component interface. Many parts of the component interface, such as the keys, are created based on settings in the referenced component.

Criteria for Setting Automatic Default Properties

To be able to set automatic defaults for fields in the new component interface, the system needs the properties to be of a specific field or page control type.

The fields should be of the following types:

- Character
- Long character
- Number
- Signed number
- Date
- Time
- Datetime

The field should be one of the following page control types:

- Edit box
- Drop-down list box
- Check box
- Radio button

The field cannot be invisible and should not be the same as the key field of the immediate parent.

Collections must have at least one child property that satisfies the field or page control criteria for providing the field by default. Collections with no properties are not added.

For a field on a secondary page to be selected for the default properties process, it must satisfy all the criteria for field type and page control and must be at the same level as the host page.

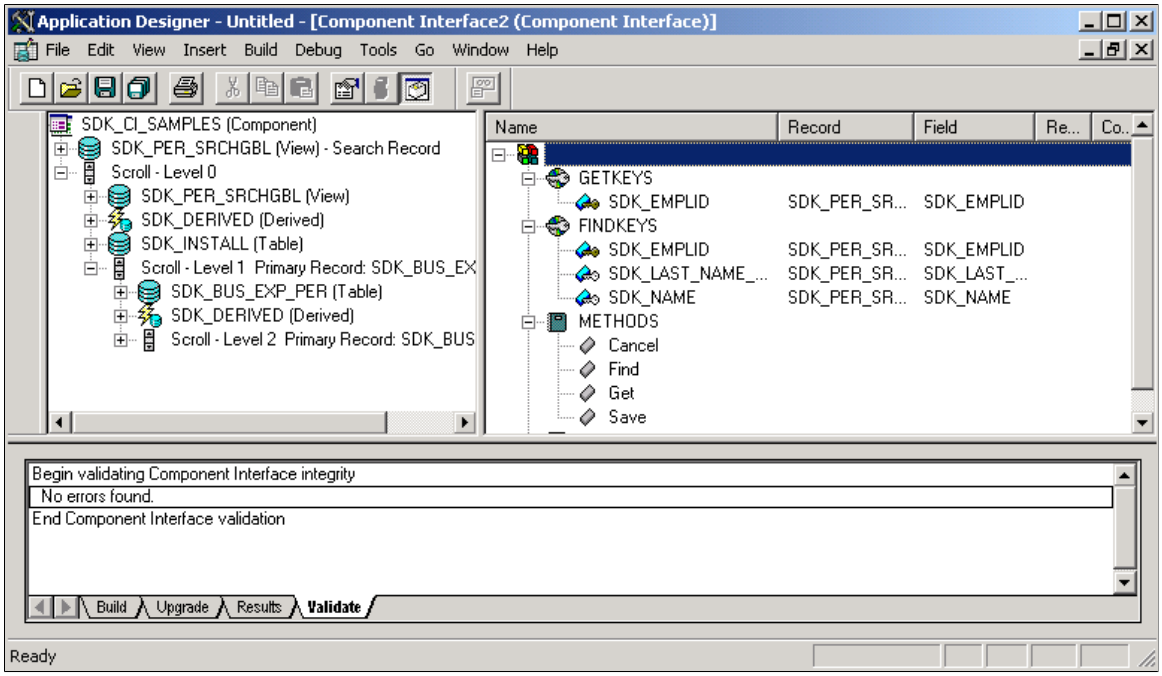
Additionally, the component tree that a component interface uses to order the properties lists the fields in the record based on their order in the record definition and not the order of the fields on the page. If the component tree lists the fields of a record based on the page, the properties of the component interface will reflect that order.

Creating New Component Interfaces

This section discusses how to create a new component interface.

Image: PeopleSoft Application Designer

This example shows a new component interface with no properties yet defined.



To create a new component interface:

1. Select File > New from the PeopleSoft Application Designer menu.
2. Select Component Interface from the New dialog box.
3. Select the component on which to base this component interface.

   After you select the appropriate component, you see a message asking whether you want the fields that are exposed in the selected component to become the default properties of the component interface.

   Note: Not all fields on the component interface can have automatic defaults created for them.

   ---

4. Click *Yes* to confirm the default property definitions or *No* if you don't want any properties initially created.

   If you elect to have the property definitions automatically provided by the system by default, then all properties that appear on the pages of the underlying component are added to the component interface. Even though the system adds the default properties, you may need to move other properties into the component view for the component to work.

   An untitled component interface appears, showing the Get keys and Find keys. Create keys are produced only if the underlying component can run in Add mode (the example preceding this procedure does not have Create keys, because the search record of the underlying component cannot run in Add mode). PeopleSoft Application Designer generates the keys for you as you drag definitions.

   The standard methods Cancel, Find, Get, and Save are automatically created. The Create method is not automatically created unless the component supports the Add mode.

Note: You can begin adding properties to a new component interface at any point. However, you cannot add any user-defined methods to the component interface until you have saved the component interface.

5. Save the component interface.

   After you have saved the component interface, you can further define user-defined methods.

Naming Component Interface Definitions

Like every other definition in PeopleTools, component interfaces must have unique names. The naming of component interfaces should be consistent and systematic. Also, the name should not be changed after the component interface is part of a production system—other applications depend on a consistent name with which to reference the component interface.

If you are changing the structure of a component interface such that an existing program can no longer access it correctly, create a new component interface rather than updating the existing one. No version property is on a component interface, so if you must create a new version of a delivered component interface, adhere to a standard naming guideline to avoid confusion. A suggested naming guideline is:

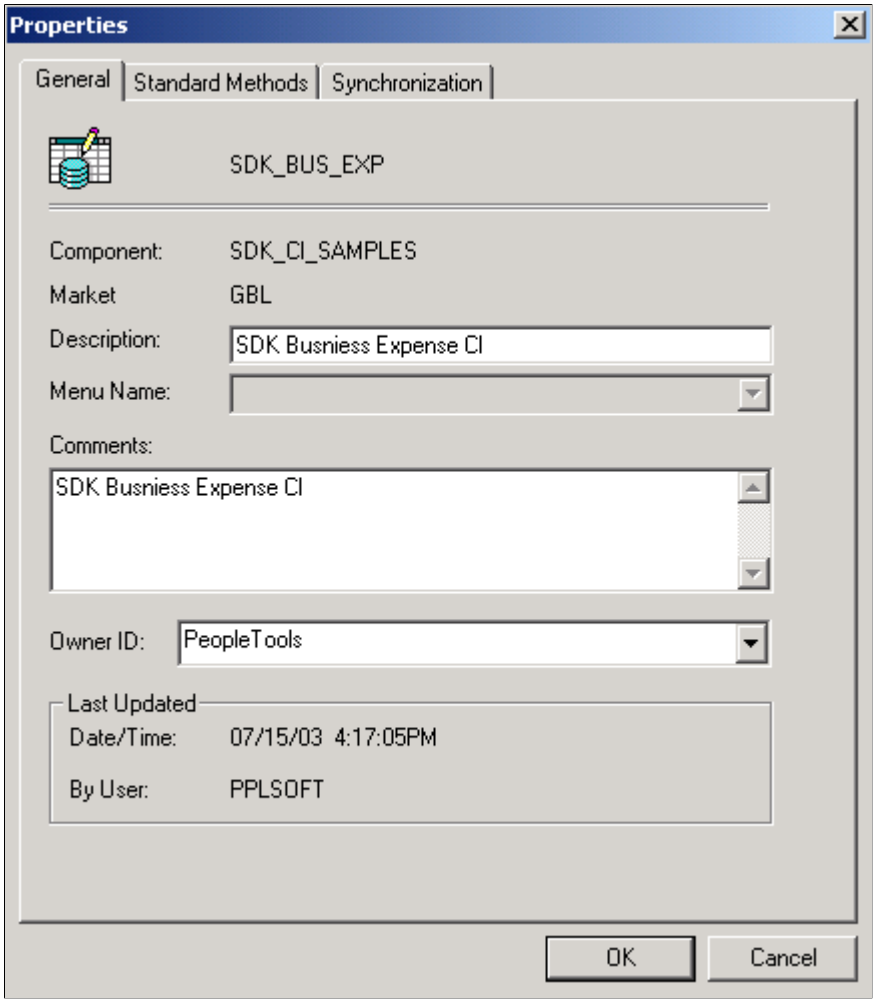
- LOCATION (original component interface).
- LOCATION_V2 (version two of the component interface).

Associating Component Interfaces with Menus

This applies to component interfaces built from components that are already attached to one or more menus.

Image: Properties – General tab

Use the Properties – General tab to associate a component interface with a menu.



To associate a component interface with a menu:

1. Select File > Open from the PeopleSoft Application Designer menu to open an existing component interface.
2. Select File > Definition Properties from the PeopleSoft Application Designer menu.

   The Definition Properties dialog box appears.
3. Select the appropriate menu name on the General tab for this component interface.

Note: Associate a menu with a component interface only when PeopleCode is in the component that uses the %Menu system variable.

Determining the Fields to Expose in Component Interfaces

You expose fields from a component in the component interface by dragging a record field or a scroll from the component view into the component interface view. However, some forethought is required before exposing a component as a component interface. You need to have a thorough understanding of the underlying component so that you expose fields that are required in the external system. For example, if

the component has a field called SSN, you need to be sure that the SSN field is required before exposing it to the external system. Expose only those properties that are necessary.

The component view displays fields that are available in the component buffer at runtime. For example, if a record containing 10 fields has only 3 fields included on a page, then the component view will list only those 3 fields.

The first time that you drag a scroll from the component view to the component interface view, the system uses the following rules to determine what properties to expose:

- Keys are exposed only at the highest-level collection in which they first appear.

  In some cases, this is not appropriate. When an effective-dated component that has the same level-zero and level-one record is exposed through a component interface, it should be exposed the same way in which it appears on a page in the component. In this case, only one key field typically appears at level zero and the effective-date keys appear at level one. The component interface wrapper should expose the page in the same fashion—removing keys that do not appear in the level-zero scroll from the component interface top-level collection and manually adding keys that appear in level-one scroll to the second-level collection.

  Typically, you do not want to expose Get keys or Create keys as properties, because these are set before a Get or Create operation and might be inadvertently changed.

- Make sure that you do not delete all the properties within the collection; that would result in an empty collection. If such empty collections exist, remove them; otherwise, they appear with *X* in the component interface view.

- If your page does not support Add mode, then you should not expose the level-zero record of the component, because it contains data that is not specific to the component interface that you are creating.

- Do not expose fields that are not visible in the component view.

  The component optimization code might eliminate unused fields from its buffers, which results in an error when that field is accessed by the component interface.

Using Keys

This section discusses how to add and delete keys.

Understanding Keys

Keys are created automatically when you create a component interface.

The following table shows the three types of component interface keys and describes the characteristics of each:

Key Type	*Key Characteristics*
Get keys	These keys automatically map to search key fields in the search record for the underlying component. You must change Get keys only if you modify the keys of the component after you create a component interface.
Find keys	These map to both search key fields and alternate search key fields in the search record for the underlying component. You can remove any Find keys based on alternate search key fields that you don't want to make available for searching.
Create keys	If the underlying component allows the Add action, then Create keys are generated for the component interface automatically. They map to fields marked as *Srch* (search) in the search record for the component (or the add search record, if one is specified).

Typically, you must manually add keys only if new search key fields are added to the underlying component after the creation of the component interface. However, you might want to modify the Find keys—either to restrict a user from searching on a particular key or to add an alternate search key that didn't exist when the component was created.

Valid Conditions for Modifying Keys

The following conditions are valid for modifying keys.

- You can add or delete a Find key if it is based on an alternate search key field.
- You can add any type of key based on a qualifying search key field in the component, if it isn't already the basis of an existing key of the same type.

  This is necessary only if a new search key field is added to the component after you create the component interface.
- You can delete any type of key if its underlying search key field meets one of these criteria:
  - It is no longer defined as a search key field.
  - It is no longer designated as a list box item.
  - It has been deleted from the component.

Note: An *X* icon precedes a name in the component interface view if the field underlying a component interface key no longer qualifies as a key. Remove keys (or any other properties) that are marked with this symbol to ensure proper operation of the component interface.

Adding and Deleting Keys

To add a key:

1. Expand the search key collection (the first collection) in the component view.
2. Drag the key to the component interface view.

To delete a key:

1. Select the key in the component interface view.
2. Press the Del key.

Setting Properties

This section provides an overview of standard properties and discusses how to work with user defined properties.

Understanding Standard Properties

Standard properties do not appear in the component interface view in PeopleSoft Application Designer. The following tables name and define the standard properties, and list the interfaces for PeopleCode, Java, and C++.

This table contains the component interface properties:

Property Name	***Description, Programming Syntax***
CreateKeyInfoCollection	Returns a collection of items that describes the Create keys. This property is read-only. Use these interfaces to call with other programming languages. • Java: IcompIntfcPropertyInfoCollection getCreateKeyInfoCollection() • C++: HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION <CI_ NAME>_GetCreateKeyInfoCollection(HPSAPI_<CI_ NAME>)
GetKeyInfoCollection	Returns a collection of items that describes the Get keys. This property is read-only. Use these interfaces to call with other programming languages. • Java: IcompIntfcPropertyInfoCollection getGetKeyInfoCollection() • C++: HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION<CI _NAME>_GetGetKeyInfoCollection(HPSAPI_<CI_ NAME>)

Property Name	***Description, Programming Syntax***
FindKeyInfoCollection	Returns a collection of items that describes the Find keys. This property is read-only. Use these interfaces to call with other programming languages. • Java: IcompIntfcPropertyInfoCollection getFindKeyInfoCollection() • C++: HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION<CI_ NAME>_GetFindKeyInfoCollection(HPSAPI_<CI_ NAME>)
GetHistoryItems	Controls whether the component interface runs in Update/ Display mode or Update/Display All mode when the underlying component is effective-dated. If GetHistory is set to true, then historical data can be retrieved but not modified. GetHistory items work in accordance with EditHistory items. The default value is False. This property is read-only. Use these interfaces to call with other programming languages. • Java: boolean getGetHistoryItems(), void setGetHistoryItems(boolean) • C++: BOOL <CI_NAME>_GetGetHistoryItems(HPSAPI _<CI_NAME>), void<CI_NAME>_SetGetHistoryItems (HPSAPI_<CI_NAME>, BOOL)
EditHistoryItems	Controls whether the component interface runs in Update/ Display All mode, Update/Display mode, or Correction mode when the underlying component is effective-dated. If EditHistory items are set to true, then historical data can be modified. EditHistory items work in accordance with GetHistory items. The default value is False. This property is read-only. Use these interfaces to call with other programming languages. • Java: boolean getEditHistoryItems(), void setEditHistoryItems(boolean) • C++: BOOL <CI_NAME>_GetEditHistoryItems (HPSAPI_<CI_NAME>), void<CI_NAME>_ SetEditHistoryItems(HPSAPI_<CI_NAME >, BOOL)

Property Name	*Description, Programming Syntax*
InteractiveMode	Controls whether to apply values and run business rules immediately, or whether items are queued and business rules are run later, in a single step. **Note:** You should use interactive mode when testing and debugging a component interface. Interactive mode in a production environment slows performance because of the number of server trips required. If you are using a component interface as part of a batch process in which thousands of rows are to be inserted, running in interactive mode may reduce performance so much on some UNIX servers that the application times out with a connection failure. The default value is False. This property is read-only. Use these interfaces to call with other programming languages. • Java: boolean getInteractiveMode(), void setInteractiveMode(boolean) • C++: BOOL <CI_NAME>_GetInteractiveMode(HPSAPI_<CI_NAME>), void<CI_NAME>_SetInteractiveMode (HPSAPI_<CI_NAME>, BOOL)
StopOnFirstError	When this property is set to True, the first error generated by the component interface halts the program. The default value is False. This property is read-only. • Java: boolean getStopOnFirstError(), setStopOnFirstError (boolean) • C++: BOOL <CI_NAME>_GetStopOnFirstError (HPSAPI_<CI_NAME>), void<CI_NAME>_ SetStopOnFirstError(HPSAPI_<CI_NAME>, BOOL)
CompIntfcName	Returns the name of the component interface class as named in PeopleSoft Application Designer. This property is read-only. • Java: String getCompIntfcName() • C++: LPTSTR <CI_NAME>_GetCompIntfcName((HPSAPI_<CI_NAME>)
ComponentName	Returns the name of the component interface class as named in PeopleSoft Application Designer. This property is read-only. • Java: boolean getComponentName() • C++: LPTSTR <CI_NAME>_GetComponentName (HPSAPI_<CI_NAME>)

Property Name	***Description, Programming Syntax***
Description	Returns the description of the component interface class as set in PeopleSoft Application Designer. This property is read-only. • Java: boolean getDescription() • C++: LPTSTR <CI_NAME>_GetDescription((HPSAPI_<CI_NAME>)
Market	Returns the Market setting of the component used to build this component interface. This property is read-only. • Java: String getMarket() • C++: LPTSTR <CI_NAME>_GetMarket((HPSAPI_<CI_NAME>)
GetDummyRows	When a new scroll is inserted on a page, that scroll is displayed even though it has no underlying data. Any scroll that is empty has one dummy row displayed with only the defaults set. This property is *True* if the dummy row is to be displayed, *False* if it is not. The default value for this property is *True*. This property is read-write. • Java: boolean getGetDummyRows(), void setGetDummyRows(boolean) • C++: BOOL <CI_NAME>_GetGetDummyRows (HPSAPI_<CI_NAME>), void<CI_NAME>_ SetGetDummyRows(HPSAPI_<CI_NAME>, BOOL)
PropertyInfoCollection	Returns a collection of items that describes a specific property. The specific properties that are available in the propertyinfocollection are listed here. This property is read-only. Use these interfaces to call with other programming languages. • Java: IcompIntfcPropertyInfoCollection getPropertyInfoCollection() • C++: HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION <CI_NAME>_GetPropertyInfoCollection(HPSAPI_<CI_NAME>)

The Component Interface Property Info Collection object (CompIntfPropInfoCollection) supports the following "CompIntfPropInfoCollection Object Properties" (PeopleTools 8.58: PeopleCode API Reference).

Property Name	*Description, Programming Syntax*
Name	This property returns the name of the object executing the property as a string. This property is read-only. • Java: String getName() • C++: LPTSTR CompIntfcPropertyInfo_GetName (HPSAPI_COMPINTFCPROPERTYINFO)
RecordName	This property returns the record name associated with the object executing the property. This property is read-only. • Java: String getRecordName() • C++: LPTSTR CompIntfcPropertyInfo_GetRecordName (HPSAPI_COMPINTFCPROPERTYINFO)
FieldName	This property returns the field name associated with the object executing the property. This property is read-only. • Java: String getFieldName() • C++: LPTSTR CompIntfcPropertyInfo_GetFieldName (HPSAPI_COMPINTFCPROPERTYINFO \)
LabelLong	This property returns the record field Long Name value as a string. If a component override exists for this value, it is not included. This property is read-only. • Java: String getLabelLong() • C++: LPTSTR CompIntfcPropertyInfo_GetLabelLong (HPSAPI_COMPINTFCPROPERTYINFO)
LabelShort	This property returns the record field ShortName value as a string. If a component override exists for this value, it is not included. This property is read-only. • Java: String getLabelShort() • C++: LPTSTR CompIntfcPropertyInfo_GetLabelShort (HPSAPI_COMPINTFCPROPERTYINFO) • COM: String LabelShort
IsCollection	This property returns True if the object executing the property is a data collection, False otherwise. If IsCollection is True, other field-oriented properties like Required, Type, Xlat, Yes, No, Prompt, and Format are undefined. If IsCollection is False, the object represents a field and all the previous properties are defined as described. This property is read-only. • Java: boolean getIsCollection() • C++: BOOL CompIntfcPropertyInfo_GetIsCollection (HPSAPI_COMPINTFCPROPERTYINFO)

Property Name	***Description, Programming Syntax***
Type	This property returns the field type, as a number, of the object. This property is read-only. • Java: long getType() • C++: PSI32 CompIntfcPropertyInfo_GetType(HPSAPI_COMPINTFCPROPERTYINFO)
OAType	This property returns the field type, as a number, of the object. This property is read-only. • Java: long getOAType() • C++: PSI32 CompIntfcPropertyInfo_GetOAType (HPSAPI_COMPINTFCPROPERTYINFO)
Format	This property returns the field format for the object executing the property (that is, name, phone, zip, SSN, and so on) as a number. This property is read-only. • Java: String getFormat() • C++: PSI32 CompIntfcPropertyInfo_GetFormat(HPSAPI_COMPINTFCPROPERTYINFO)
Key	This property returns *True* if the object executing the property is a key, *False* otherwise. This property is read-only. • Java: boolean getKey() • C++: BOOL CompIntfcPropertyInfo_GetKey (HPSAPI_COMPINTFCPROPERTYINFO hCompIntfcPropertyInfo)
Required	This property returns *True* if the object executing the property is a required property, *False* otherwise. This property is read-only. • Java: boolean getRequired() • C++: BOOL CompIntfcPropertyInfo_GetRequired (HPSAPI_COMPINTFCPROPERTYINFO)
Xlat	This property returns *True* if the object executing the property is associated with an XLAT table, *False* otherwise. This property is read-only. • Java: String getXlat() • C++: BOOL CompIntfcPropertyInfo_GetXlat(HPSAPI_COMPINTFCPROPERTYINFO)

Property Name	*Description, Programming Syntax*
Yesno	This property returns *True* if the object executing the property is associated with the Yes/No table, *False* otherwise. This property is read-only. • Java: boolean getYesno() • C++: BOOL CompIntfcPropertyInfo_GetYesno(HPSAPI_COMPINTFCPROPERTYINFO)
Prompt	This property returns *True* if the object executing the property is associated with a prompt table, *False* otherwise. This property is read-only. • Java: boolean getPrompt() • C++: BOOL CompIntfcPropertyInfo_GetPrompt (HPSAPI_COMPINTFCPROPERTYINFO)
Length	This property returns the length of the object executing the property. This property is read-only. • Java: long getLength() • C++: PSI32 CompIntfcPropertyInfo_GetLength(HPSAPI_COMPINTFCPROPERTYINFO)
DecimalPosition	This property returns the decimal position for the object executing the property. This property is read-only. • Java: long getDecimalPosition() • C++: PSI32 CompIntfcPropertyInfo_GetDecimalPosition (HPSAPI_COMPINTFCPROPERTYINFO)
IsReadOnly	This property returns *True* if the property marked read-only in the component interface definition; *False* otherwise. This property is read-only. • Java: boolean getIsReadOnly() • C++: BOOL CompIntfcPropertyInfo_GetIsReadOnly (HPSAPI_COMPINTFCPROPERTYINFO)
Altkey	This property returns *True* if the object executing the property is an alternate key, *False* otherwise. This property is read-only. • Java: boolean getAltkey() • C++: BOOL CompIntfcPropertyInfo_GetAltkey(HPSAPI_COMPINTFCPROPERTYINFO)
Listboxitem	This property returns *True* if the object executing the property is associated with a list box, *False* otherwise. This property is read-only. • Java: boolean getListboxitem() • C++: BOOL CompIntfcPropertyInfo_GetListboxitem (HPSAPI_COMPINTFCPROPERTYINFO)

Example of PropertyInfoCollection

Here is a Java example that calls PropertyInfoCollection:

```
IcompIntfcPropertyInfoCollection oLO_PropInfoColl
IcompIntfcPropertyInfo oLO_PropInfoItem

oLO_PropInfoColl = oCI.getPropertyInfoCollection();
for (int I=0; I < oLO_PropInfoColl.getCount(); I++) {
   oLO_PropInfoItem = oLO_PropInfoColl.item(i);

   System.out.println("\t Name = " + oLO_PropInfoColl.getName());
   System.out.println("\t Record Name = " + oLO_PropInfoColl.getRecordName());
   System.out.println("\t Field Name = " + oLO_PropInfoColl.getFieldName());
   System.out.println("\t Label Long = " + oLO_PropInfoColl.getLabelLong());
   System.out.println("\t Label Short = " + oLO_PropInfoColl.getLabelShort());
   System.out.println("\t IsCollection = " + oLO_PropInfoColl.getIsCollection());
   System.out.println("\t Type = " + oLO_PropInfoColl.getType());
   System.out.println("\t OAType = " + oLO_PropInfoColl.getOAType());
   System.out.println("\t Format = " + oLO_PropInfoColl.getFormat());
   System.out.println("\t Is Get Key? = " + oLO_PropInfoColl.getKey());
   System.out.println("\t Is Required = " + oLO_PropInfoColl.getRequired());
   System.out.println("\t Is Xlat? = " + oLO_PropInfoColl.getXlat());
   System.out.println("\t Is Yesno? = " + oLO_PropInfoColl.getYesno());
   System.out.println("\t Prompt = " + oLO_PropInfoColl.getPrompt());
   System.out.println("\t Length = " + oLO_PropInfoColl.getLength());
   System.out.println("\t DecimalPosition = " + oLO_PropInfoColl.
      getDecimalPosition());
   System.out.println("\t Is Read Only? = " + oLO_PropInfoColl.
      getIsReadOnly());
   System.out.println("\t Is Alt Key? = " + oLO_PropInfoColl.getAltkey());
   System.out.println("\t Is ListBox item? = " + oLO_PropInfoColl.
      getListboxitem());
```

Object Adapter

The name of the property is OAType, and it holds the value of the object adapter type. Exposing this property and supplying the associated methods enables you to detect possible data type mismatches between the database and the component interface object.

The Java methods are:

getOAType()	Returns the object adapter type.
getType()	Returns the type of the property of a particular database field.

For example:

```
public static void printPropertyType(String propName, ICompIntfcPropertyInfo iPrope⇒
rtyInfo) {

String strOAType = null;
String strDBType = null;

    try {
        switch ((int)iPropertyInfo.getOAType()) {
            /* Object Adapter Type == 0 */
            case CIPropertyTypes.PSPROPERTY_OA_TYPE_BOOL:
                    strOAType = "BOOL";
                    break;
             /* Object Adapter Type == 1 */
             case CIPropertyTypes.PSPROPERTY_OA_TYPE_NUMBER:
                    strOAType = "INTEGER";
                    break;
             /* Object Adapter Type == 2 */
             case CIPropertyTypes.PSPROPERTY_OA_TYPE_FLOAT:
                    strOAType = "FLOAT";
```



```
                break;
             /* Object Adapter Type == 3 */
             case CIPropertyTypes.PSPROPERTY_OA_TYPE_STRING:
                strOAType = "STRING";
                break;
            }

        switch ((int)iPropertyInfo.getType()) {
            /* Database Type == 0 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_CHARACTER:
                strDBType = "CHARACTER";
                break;
            /* Database Type == 1 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_LONG_CHARACTER:
                strDBType = "LONG_CHARACTER";
                break;
            /* Database Type == 2 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_NUMBER:
                strDBType = "NUMBER";
                break;
            /* Database Type == 3 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_SIGNED_NUMBER:
                strDBType = "SIGNED NUMBER";
                break;
            /* Database Type == 4 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_DATE:
                strDBType = "DATE";
                break;
            /* Database Type == 5 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_TIME:
                strDBType = "TIME";
                break;
            /* Database Type == 6 */
            case CIPropertyTypes.PSPROPERTY_DB_TYPE_DATETIME:
                strDBType = "DATETIME";
                break;
            }

        }
        catch (Exception e) {
            e.printStackTrace();
        }

        System.out.println("\n" + propName +
             " Object Adapter Type is:  " + strOAType +
             ", Database Type is:  " + strDBType);
    }
```

Component Interface Collection Property

This table describes the component interface collection property Count.

Name	***Description, Programming Syntax***
Count	Returns the number of items in a collection. • Java: long getCount() • C++: PSI32 CompIntfcCollection_GetCount(HPSAPI_<CI_NAME>)

Data Item Property

This table describes the data item property ItemNum:

Name	*Description, Programming Syntax*
ItemNum	Returns the position of the row within the given collection of a DataRow. • Java: long getItemNum() • C++: PSI32 <CI_NAME>_GetItemNum(HPSAPI_<CI_NAME>)

Note: The component interface classes contain information about PropertyInfo properties and related PeopleCode.

Creating User-Defined Properties

User-defined properties are those properties on the underlying component that are exposed through the component interface. User-defined properties are derived from the component to which the component interface is associated and must be added manually. They are the specific record fields that you expose to an external system with the component interface. You create user-defined properties in addition to the standard properties to enable data manipulation of the component. When you create a new component interface, if you accept the default properties, user-defined properties are created automatically for each field displayed to the user on the underlying component.

User-defined properties are the points where the component and the underlying database are exposed to the external system. This is the means that component interfaces use to add or change fields and data in the database.

To create a user-defined property, drag a record, field, or scroll from the component view to the component interface view.

Where you insert the definition in the component interface view does not matter. When the component interface is opened, the system automatically converts the field or record into a component interface property and places it in the appropriate place in the list of properties. Also, when you drag a definition from the component view into the component interface view, all child definitions are brought into the component interface automatically. After these child properties are added to the component interface, you can remove each property individually, if desired.

Dragging a key from the search records, which precede the level-zero record in the page view, adds a key to all appropriate key collections (Get, Create, and Find) in the component interface. Because appropriate keys are added automatically when a component interface is first created, you typically must add keys only if the new keys are added to the underlying component after the creation of the component interface.

Deleting User-Defined Properties

To delete a property:

1. Select the property to be deleted.
2. Either press the Del key on the keyboard, or right-click the key and select Delete.

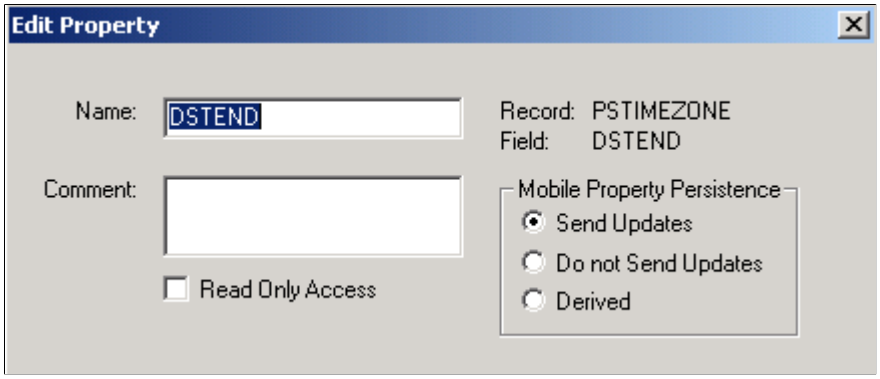
Standard Windows behavior is employed for selecting multiple properties using the Shift and Ctrl keys.

Renaming User-Defined Properties

Property names are automatically generated according to the corresponding fields from the component. If these names are cryptic, you might want to rename these properties to explain them better. Renaming a property does not change the field that the property references.

Image: Edit Property dialog box

This example shows the Edit Property dialog box. Use this dialog box to rename user-defined properties for component interfaces.



Important! PeopleSoft Mobile Agent is a deprecated product. The options listed in the Mobile Property Persistence group box of the Edit Property dialog box exist for backward compatibility only.

To rename a property:

1. Double-click the property name or right-click the property name and select Edit Name from the menu.
2. Enter the new property name.

   Programs accessing this component interface must reference the new property name. For example, if SDK_NAME was changed to NAME, programs must use NAME instead of SDK_NAME.
3. Add any comments that might be helpful.
4. Select the Read-Only check box to make this property read-only.
5. If this property is for a mobile application, select a radio button that sets the persistence of the property.

   - Send Updates is the default behavior for a mobile property.

     Any changes or additions to this property on a mobile instance are synchronized to the server.
   - If a mobile property is set to Do not Send Updates, this property is not synchronized up to the server, but the value is maintained on the device.

- A Derived property is used only at mobile runtime. Any values that are set or added to this property exist only for the runtime life of the object. No persistence of this data on the device exists, so it is subsequently never uploaded to the server.

Note: PeopleSoft Application Designer generates an error message if it detects that a component interface has properties that resolve to the same name when creating, saving, or opening a given component interface.
For example, NAME1 and NAME_1 both resolve to the same name when PeopleSoft APIs are built. The set and get functions that are generated for the properties RTE_CNTL_TYPE1 and RTE_CNTL_TYPE_1 are: public String getRteCntlType1(), and public void setRteCntlType1(String inRteCntlType1). To fix this condition, name the properties so that they do not resolve to the same name.

Creating Reference Properties

Each component interface is isolated and unaware of the other component interfaces in the system. To access and update information from other component interfaces, references establish relationships between component interfaces.

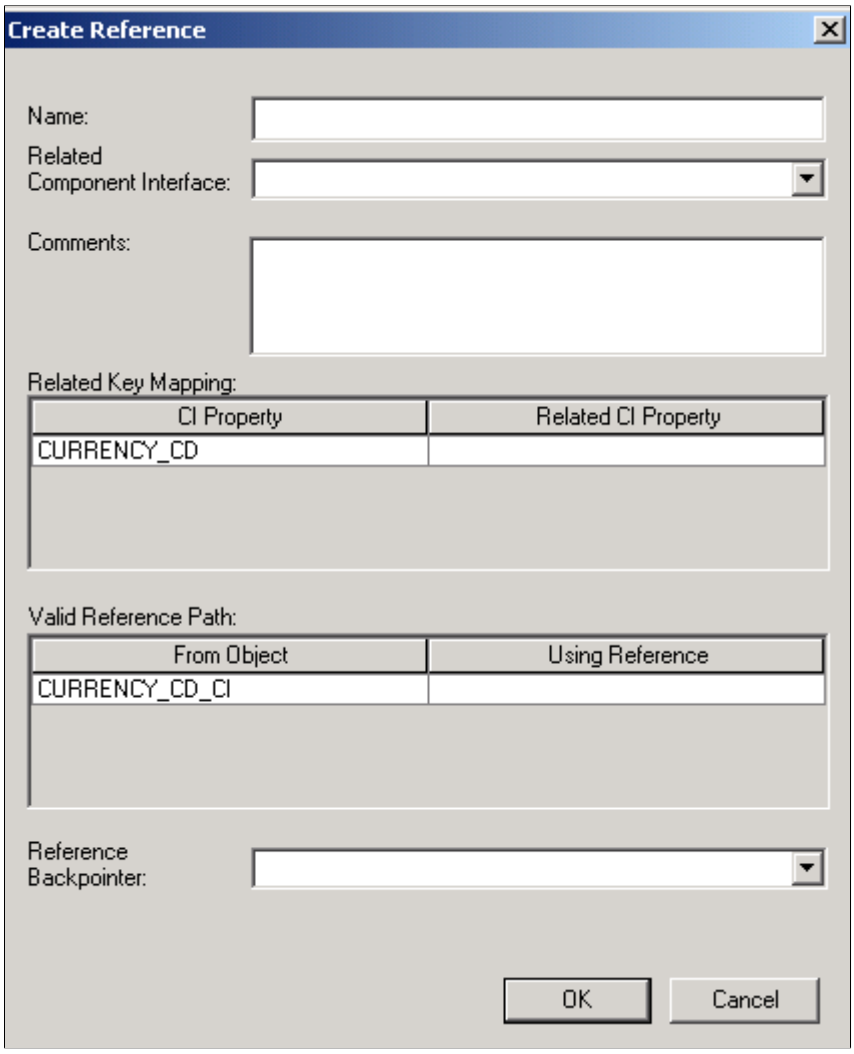
Create a reference property in one component interface to access data exposed in another component interface. For example, the Customer object and the component interface exposing its properties include properties such as the customer's name, address, and telephone numbers. Another object, Contact, includes data associated with all contacts in the system. The link between a specific customer and its associated contacts is owned by the Contact record, not the Customer record.

Therefore, to access contact data, the Customer component interface needs a reference property referring to the Contact component interface. For you to update contact data from the Customer component interface, the reference must include a valid reference path and reference backpointer to the customer ID.

Access the Create Reference dialog box by right-clicking the property and selecting Create Reference.

Image: Create Reference dialog box

This example illustrates the fields and controls on the Create Reference dialog box. You can find definitions for the fields and controls later on this page.



The Create Reference dialog box has the following fields:

Name	Describes the name of the reference you are creating.
Related Component Interface	Designates the component interface referenced from the current component interface.
Comments	Enter any comments to track the reference.
Related Key Mapping	Maps the property from the related component interface to the selected component interface property.
Valid Reference Path	Supports dynamic enumeration of the objects that can be selected as the value of the reference property being defined. This effectively filters these values so that you can select only objects that support the defined reference.

	Because references use the concept of a *walkpath* to go from level zero of one component interface to level zero of another component interface, and then "walk" down to the lower levels of the component interface, only the level zero references are displayed in the Valid Reference Path drop-down list of a reference definition.
Reference Backpointer	Refers to the path back to the original component interface.

Making Properties Read-Only

You can make any property read-only. At runtime, the value of a read-only property can be read but not updated.

To make a property read-only:

1. Select the property.
2. Select Edit > Toggle Read Only Access from the PeopleSoft Application Designer menu.

   A *Y* appears in the Read Only column of the component interface view corresponding to each property that you selected to be read-only.

Note: You can double-click the icon of any existing user-defined property to edit its name or comment or to toggle read-only access.

Working with Collections

A collection is a property that points to a scroll, rather than a field, in the underlying component for a component interface. A collection groups multiple fields in a scroll. All the fields in the scroll are mapped to a property. These properties are part of the collection.

You create collections the same way you create properties—drag the scroll from the component view into the component interface view. Consider these points when creating collections:

- When you drag a scroll into the component interface view, all child scrolls come with it.

  This is the same behavior that you would expect when creating a property. Child properties are always added automatically when you drag a field from the component view to the component interface view. After the property or collection has been created, you can delete individual child properties or collections manually, if necessary.

- When you drag a scroll into the component interface view, all record fields contained in that scroll come with it—not just those from the record that defines the scroll.

  The fields from all records at that scroll level are exposed as part of the same collection.

- Keys that appear in parent and child scrolls are not added to child collections.

  For the component interface to function as expected, the keys must remain synchronized at all levels of the component. Having keys at lower levels makes compromising this synchronization possible.

Therefore, lower-level keys are not introduced into the component interface and are not exposed to the user because those keys have already been set at the parent level.

- When you drag a child scroll into the component interface view, parent collections are created automatically.

  For example, if you drag just the level-two scroll from the component view into the component interface view, a level-zero collection and a level-one collection are created automatically in the component interface. This hierarchy of collections is necessary so that you can navigate to the child collection at runtime.

Working with Methods

This section provides and overview of session functions and methods, standard methods and collection methods.

Understanding Session Functions and Methods

The session functions and methods connect to a session on an Application server. This connection must be made before you can use the component interface methods.

Component Interface Session Functions

This table describes the component interface session function createSession:

Name	***Description, Programming Syntax***
createSession (In PeopleCode, &session = %session)	Returns a session object. • Java: ISession API.createSession() • C++: HPSAPI_SESSION PSApiCreateSession()

Component Interface Session Methods

This table describes the component interface session methods:

Name	***Description, Programming Syntax***
Connect (not used in PeopleCode)	Connects to the application server. Use these interfaces to call with other programming languages. • Java: boolean connect(long apiVersion, string server, string username, string password, byte[ ] ExternalAuth) • C++: Bool session_Connect(HPSAPI hSession, PSI32 ApiVersion, LPTSTR server, LPSTR username, LPTSTR password, PSAPIVAPBLOB ExternalAuth)

Name	*Description, Programming Syntax*
getCompIntfc	Returns a reference to a component interface. getCompIntfc also checks to determine whether the given user that is connecting has the appropriate security to access the component interface. Use these interfaces to call with other programming languages. • Java: I<CI_Name> getCompIntfc(string ciName) • C++: HPSAPI_<CI_Name> Session_GetCompIntfc (HPSAPI_SESSION hsession, LPTSTR ciName)

Understanding Standard Methods

A method is a definition that performs a specific function on a component interface at runtime. Each standard method is added by default when the component interface is created and is available in PeopleCode and other programming languages. Like properties, methods are saved as part of a component interface definition. Two main types of methods are available: standard methods and user-defined methods.

Standard Methods	*Description, Programming Syntax*
Cancel	Backs out of the current component interface, canceling any changes made since the last save. This is equivalent to clicking the Return to Search button online. Returns True on success, and False on failure. Use these interfaces to call with other programming languages. • Java: boolean cancel() • C++: BOOL <CI_NAME>_Cancel(HPSAPI_<CI_NAME> hObj)
Create	Creates a new instance of a component interface. This is equivalent to creating a new record in Add mode online. Returns True on success, and False on failure. Use these interfaces to call with other programming languages. • Java: boolean create() • C++: BOOL <CI_NAME>_Create(HPSAPI_<CI_NAME> hObj)

Standard Methods	*Description, Programming Syntax*
Find	Performs a partial key search for a particular instance of a component interface, using the search keys at level 0. Returns a collection of component interface instances which match the search criteria. If no component interface instances match the search criteria, the count on the collection is zero. Use these interfaces to call with other programming languages. • Java: <CI_NAME>Collection find() • C++: HPSAPI_<CI_NAME>COLLECTION <CI_ NAME>_Find(HPSAPI_<CI_NAME> hObj)
Get	Retrieves a particular instance of a component interface. This is equivalent to opening a record in Update/Display or Correction mode when online with a PeopleSoft application. Returns True on success, and False on failure. Use these interfaces to call with other programming languages. • Java: boolean get() • C++: BOOL <CI_NAME>_Save(HPSAPI_<CI_NAME> hObj)
Save	Saves an instance of a component interface. This is equivalent to clicking the Save button in the online system. Returns True on success, and False on failure. You should cancel after a save. Use these interfaces to call with other programming languages. • Java: boolean save() • C++: BOOL <CI_NAME>_Save(HPSAPI_<CI_NAME> hObj)
GetPropertyByName	Returns the value of a property that is specified by name. This function typically is used only in applications that cannot get the names of the component interface properties until runtime. Use these interfaces to call with other programming languages. • Java: Object getPropertyByName(String str) • C++: HPSAPI_OBJECT <CiCollectionItem>_ GetPropertyByName(HPSAPI_<CI_COLLECTION_ ITEM> hColItem, LPTSTR Name)

Standard Methods	*Description, Programming Syntax*
SetPropertyByName	Sets the value of a property that is specified by name. This function typically is used only in applications that cannot set the names of the component interface properties until runtime. Use these interfaces to call with other programming languages. • Java: long setPropertyByName(String str, Object o) • C++: PSI32 <CiCollectionItem>_SetPropertyByName (HPSAPI_<CI_ COLLECTION_ITEM> hColItem, LPTSTR name, HPSAPI_OBJECT Value)
GetPropertyInfoByName (In PeopleCode, CompIntfPropInfoCollection)	Returns specific information, such as length, about the definition of a property that is specified by name. This function typically is used only in applications that cannot get the names of component interface properties until runtime or by applications that need to provide a dynamic list of values that would normally be found in prompt tables. Use these interfaces to call with other programming languages. • Java: IcompIntfcPropertyInfo getPropertyInfoByName (String name) • C++: HPSAPI_ COMPINTFCPROPERTYINFO<CiPropOrItem>_ GetPropertyInfoByName(HPSAPI_<CIPROPORITEM> hPropOrItem, LPTSTR name) where CiPropOrItem is the name of either a property or an item in a collection.

By default, each component interface is created with four standard methods—Cancel, Find, Get, and Save. Additionally, the Create standard method is generated if Create keys have been added to the component interface.

Example for GetPropertyInfoByName

The GetPropertyInfoByName method returns an object containing the property information. Here is a Java example that calls GetPropertyInfoByName:

```
IcompIntfcPropertyInfo oCompIntfcPropertyInfo
oCompIntfcPropertyInfo = oCI.getPropertyInfoByName(tempName);
System.out.println(oCompIntfcPropertyInfo.getName());
if (!oCompIntfcPropertyInfo.getIsCollection()) {
  System.out.println("\t Format = " + oCompIntfcPropertyInfo.getFormat());
    System.out.println("\t Type = " + oCompIntfcPropertyInfo.getType());
}
System.out.println("\t Is Required = " + oCompIntfcPropertyInfo.
   getRequired());
System.out.println("\t Is Collection? = " + oCompIntfcPropertyInfo.
   getIsCollection ());
System.out.println("\t Is Read Only? = " + oCompIntfcPropertyInfo.
   getIsReadOnly());
System.out.println("\t Is Get Key? = " + oCompIntfcPropertyInfo.getKey());
System.out.println("\t Label Long = " + oCompIntfcPropertyInfo.
   getLabelLong());
System.out.println("\t Label Short = " + oCompIntfcPropertyInfo.
   getLabelShort());
System.out.println("\t Length = " + oCompIntfcPropertyInfo.getLength());
System.out.println("\t Name = " + oCompIntfcPropertyInfo.getName());
```



```
System.out.println("\t Is Xlat? = " + oCompIntfcPropertyInfo.getXlat());
System.out.println("\t Is Yesno? = " + oCompIntfcPropertyInfo.
   getYesno());
```

Note: When creating a new component interface, you must save the component interface before the standard methods are created. PeopleSoft Application Designer adds the standard methods upon the first save of a new component interface.

Related Links

"CompIntfPropInfoCollection Object Properties" (PeopleTools 8.58: PeopleCode API Reference)

Understanding Collection Methods

The first item in a component interface collection is always indexed as item 1 from PeopleCode, which is consistent with other PeopleCode processing. From Java and C++ programs, this item is indexed as item 0.

Component Interface Collection Properties

This table describes the component interface collection properties:

Data Collection Method	***Action, Usage***
Count	Returns the number of items in a collection. Use these interfaces to call with other programming languages. • Java: long getCount() • C++: PSI32 <CiCollectionName>_GetCount (HPSAPI_ <CI_COLLECTION_NAME> hCol)
ItemByName (not used in PeopleCode)	Returns the property in the collection. It takes *Name* as a parameter. Use these interfaces to call with other programming languages. • Java: ICompIntfcPropertyInfo itemByName(String Name) • C++: CompIntfcPropertyInfoCollection _ItemByName (HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION, LPTSTR Name)

Data Collection Method	*Action, Usage*
InsertItem(Index)	Inserts a new item. This is equivalent to clicking the Add button to insert a new row when online. It takes *Index* as a parameter and follows the same conventions for performing business rules (PeopleCode) as the online system. Use these interfaces to call with other programming languages. • Java: <CiCollectionName> insertItem(long Index) • C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_InsertItem(HPSAPI_<CI_ COLLECTION_NAME> hCol, PSI32 Index)
DeleteItem(Index)	Deletes the item that is designated by *Index.* This is equivalent to clicking the Delete button to delete the selected row when online. Use these interfaces to call with other programming languages. • Java: boolean deleteItem(long Index) • C++: BOOL <CiCollectionName>_DeleteItem(HPSAPI_ <CI_COLLECTION_NAME> hCol, PSI32 Index)
Item(Index)	Takes an item number as a parameter and returns a definition of the type that is stored in the specified row in the collection. For example, if the collection is a data collection, the return value is a DataRow. If the collection value is a PropertyInfoCollection, then the return value is a PropertyInfo definition, and so on. Use these interfaces to call with other programming languages. • Java: <CiCollectionName> item(long Index) • C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_Item(HPSAPI_<CI_ COLLECTION_NAME> hCol, PSI32 Index) (HPSAPI_ COMPINTFCPROPERTYINFOCOLLECTION, PSI32)
ItemByKeys(keys)	Identifies and finds a specific item, based on keys. The keys vary according to the design of the collection. Use these interfaces to call with other programming languages. • Java: <CiCollectionName> itemByKeys(String Key1, String Key2, …) • C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_ItemByKeys (HPSAPI_<CI_ COLLECTION_NAME > hCol, LPTSTR Key1, LPTSTR Key2, …)

Data Collection Method	*Action, Usage*
CurrentItem	Returns the current effective DataRow in the collection. The behavior is consistent with effective date rules that are used online. This method works with effective-dated records only. Use these interfaces to call with other programming languages. • Java: <CiCollectionName>currentItem() • C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_ CurrentItem(HPSAPI_<CI_ COLLECTION_NAME> hCol)
CurrentItemNum (CurrentItemNumber)	Returns the item number of the current effective DataRow in the collection. The behavior is consistent with effective date rules that are used online. This method works with effective-dated records only. Use these interfaces to call with other programming languages. • Java: long currentItemNum() • C++: PSI32 <CiCollectionName>_ _CurrentItemNum (HPSAPI_<CI_COLLECTION_NAME> hCol)
GetEffectiveItem(DateString, SeqNum)	Returns the DataRow that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItem function, which returns the definition that is effective at this moment. This method works with effective-dated records only. Use these interfaces to call with other programming languages. • Java: <CiCollectionName> getEffectiveItem(String Date, long SeqNum) • C++: HPSAPI_<CI_COLLECTION_ITEM> <CiCollectionName>_ GetEffectiveItem(HPSAPI_<CI _COLLECTION_NAME> hCol, LPTSTR Date, PSI32 SeqNum)

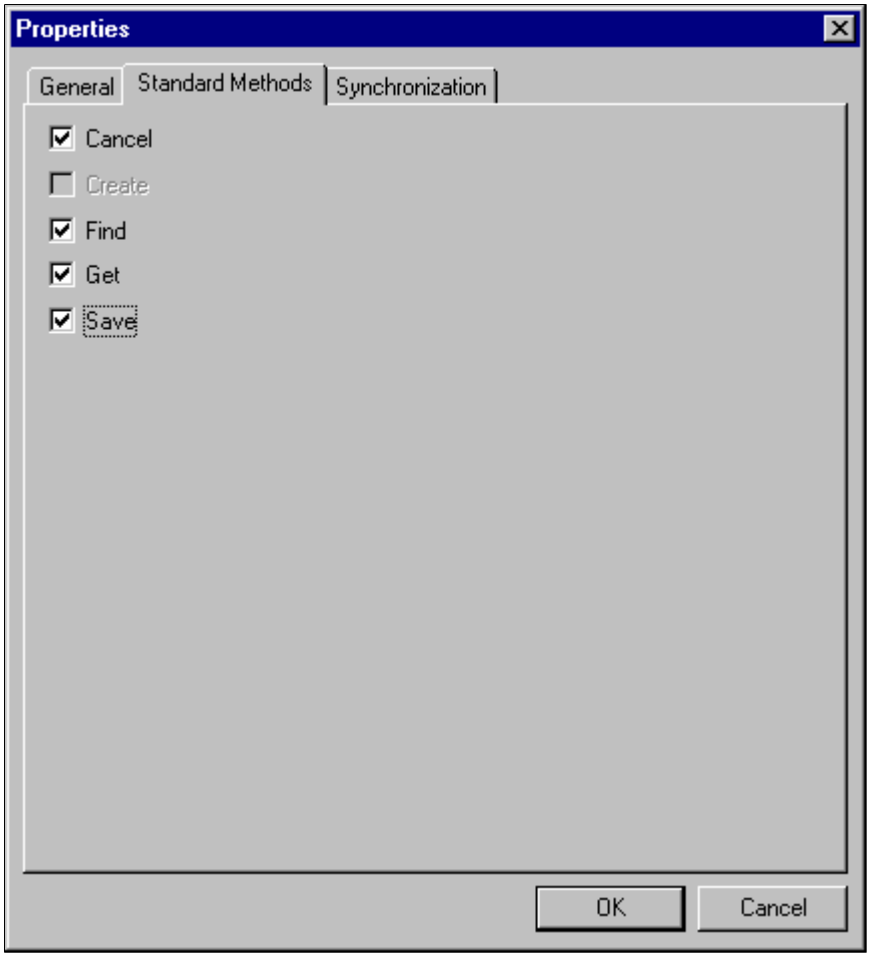
Data Collection Method	*Action, Usage*
GetEffectiveItemNum(DateString, SeqNum)	Returns the item number of the DataRow in the collection that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItemNum function, which returns the number of the definition that is effective at this moment. This method works with effective-dated records only. Use these interfaces to call with other programming languages. • Java: long getEffectiveItemNum(string Date, long SeqNum) • C++: <CiCollectionName>_GetEffectiveItemNum (HPSAPI_<CI_COLLECTION_NAME> hCol, LPTSTR Date,PSI32 SeqNum)

Enabling and Disabling Standard Methods

You can control whether standard methods are accessible at runtime.

Image: Properties – Standard Methods tab

Use the Properties – Standard Methods tab to enable or disable standard methods.



To enable or disable standard methods:

1. Select File > Definition Properties from the PeopleSoft Application Designer menu.

   The Definition Properties dialog box appears.

2. Select the Standard Methods tab.

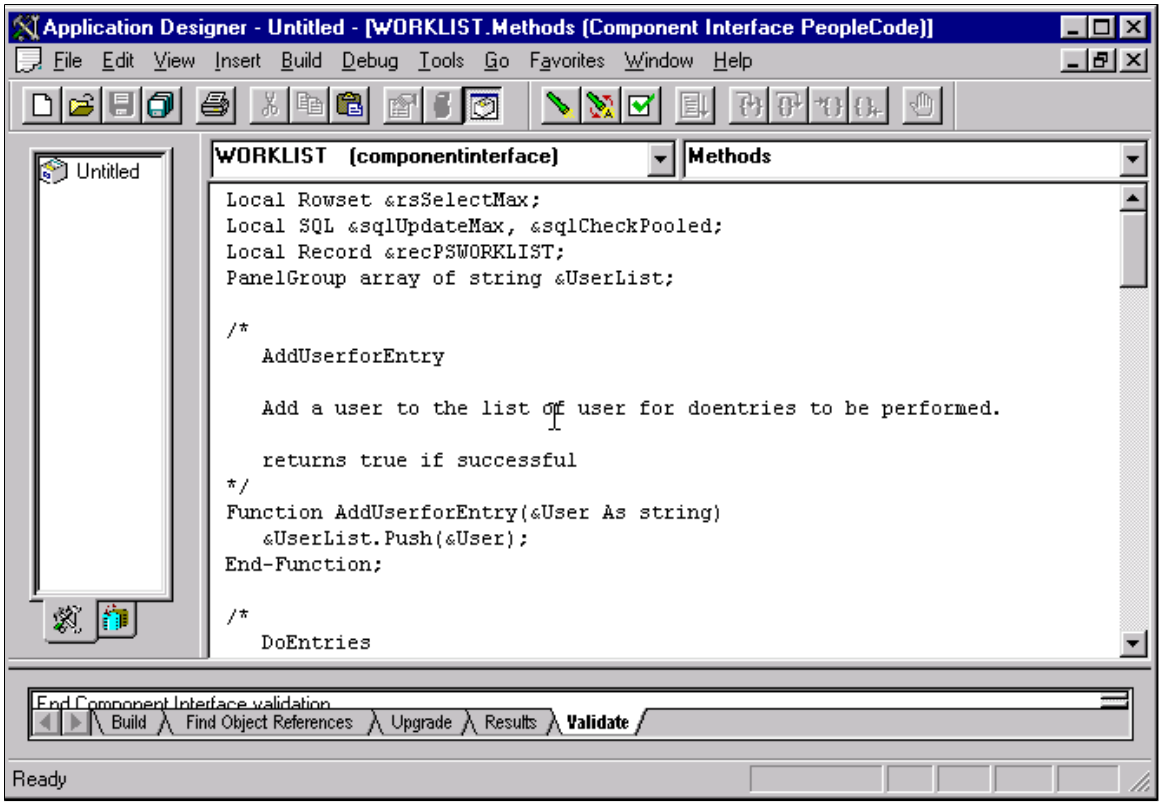
   You can enable or disable any of the standard methods selecting the corresponding check box. Doing so determines whether the method is available at runtime when the component interface is accessed. The Create option is available only if the component interface has Create keys.

Creating User-Defined Methods

This section discusses how to create user-define methods.

Image: PeopleSoft Application Designer

This example shows the work area in PeopleSoft Application Designer for creating user-defined methods.



To create a user-defined method:

1. Right-click anywhere in the component interface view.
2. Select View PeopleCode from the menu.

   The PeopleCode editor appears. If you are using a new component interface, no PeopleCode will appear in the editor because no user-defined methods have been created.
3. Write the required PeopleCode functions.

   PeopleCode functions that you write are stored in a single PeopleCode program that is attached to the component interface and associated with the Methods event.

   Note: New user-defined methods do not appear in the list of methods until you save the component interface. Double-click the icon of any existing user-defined method to return to this PeopleCode program.

   ---
4. Set permissions for the methods that you created.

   You must set permissions for every user-defined method. If you set permission to Full Access, at runtime that function is exposed to external systems as a method on the component interface object.

Note: User-defined method names must not be named *GetPropertyName*. The C header for Component Interfaces creates functions with that name so you can access each property. If you create your own *GetPropertyName* functions, you receive errors at runtime. User-defined methods can take only simple types of arguments (such as number, character, and so on) because user-defined methods can be called from Java or C/C++ as well as from PeopleCode. PeopleCode can use more complex types (like rowset, array, record, and so on), but these types of arguments are unknown to Java and C/C++.

Related Links

"Life Cycle of a Component Interface" (PeopleTools 8.58: PeopleCode API Reference)

Exporting User-Defined Methods

If you want a user-defined component interface to be exportable, meaning used by code that instantiates the component interface, the method PeopleCode definition must include a Doc statement. It is in the form of:

```
Doc <documentation for method>
```

where <documentation for method> describes what the method does

For example, the following method returns true if *foo* is positive; otherwise, it returns false.

```
Function MyFooBar(int foo) returns boolean
Doc
if (foo >0) then
return True;
else
return False;
end-if;
end-function;
```

If a component interface method is to be exposed in a web service, the Doc statement should describe the standard method after which it will be called and show an indication of each type of input parameter it requires. In the following example, the SetPassword method on the USER_PROFILE component interface has been exposed to a web service. The Doc statement in this case has a string following the Doc keyword and consists of comma-separated values: the method name Get, a string containing the new password, and another string for the confirmation password.

```
Function SetPassword(&password As string, &passwordConfirm As string) Returns boole⇒
an
   Doc "GET, NewPasswd, ConfirmPasswd
```

Validating Component Interfaces

Validation ensures that the structure of a component interface is still valid. Over time, the structure of a component interface can become invalid due to component structural changes and modifications. For example, this can happen whenever a component deletes or adds a record or field. It can also happen if the keys on the component are added or removed. Properties and keys that no longer synchronize with their associated components are marked with an X icon.

Note the following points about validating component interfaces:

- PeopleSoft Application Designer validates each component interface upon its creation. It enables validation of all component interfaces residing in the respective database.
- The validation process determines only whether the underlying component of a component interface has changed. It does not validate the PeopleCode that is associated with a component interface. To validate the PeopleCode, open the component and select Tools >Validate from the PeopleSoft Application Designer menu.
- If a component interface definition becomes invalid, you cannot save changes to it in PeopleSoft Application Designer.
- If a component interface definition is associated with an active Integration Broker service, you cannot delete it.

To correct an invalid component interface, you might have to delete properties for which appropriate fields or records no longer exist. If the structure of the source component has changed, you might have to delete old properties and re-add the new properties in their appropriate locations. You may also need to rename a property or collection.

To validate a component interface:

1. Open the component interface in PeopleSoft Application Designer.

   Validation occurs automatically whenever you open a component interface in PeopleSoft Application Designer.

2. Select Tools > Validate for Consistency from the PeopleSoft Application Designer menu to validate an open component interface.

   Whenever you change components or other related definitions, you should validate a component interface through PeopleSoft Application Designer.

To validate multiple component interfaces:

1. Select Tools, Validate Component Interfaces in PeopleSoft Application Designer.

   CI Validator dialog box appears.

2. Click the Find CIs button to list out all the component interfaces available in the database.
3. Select the component interfaces to validate. Use the Select All button to validate all the listed component interfaces.
4. Click the Validate CIs button to perform the validation. The Results box displays the success status of the validations and lists out any failed validation.

   The detailed result is displayed in the Validate tab in Application Designer.

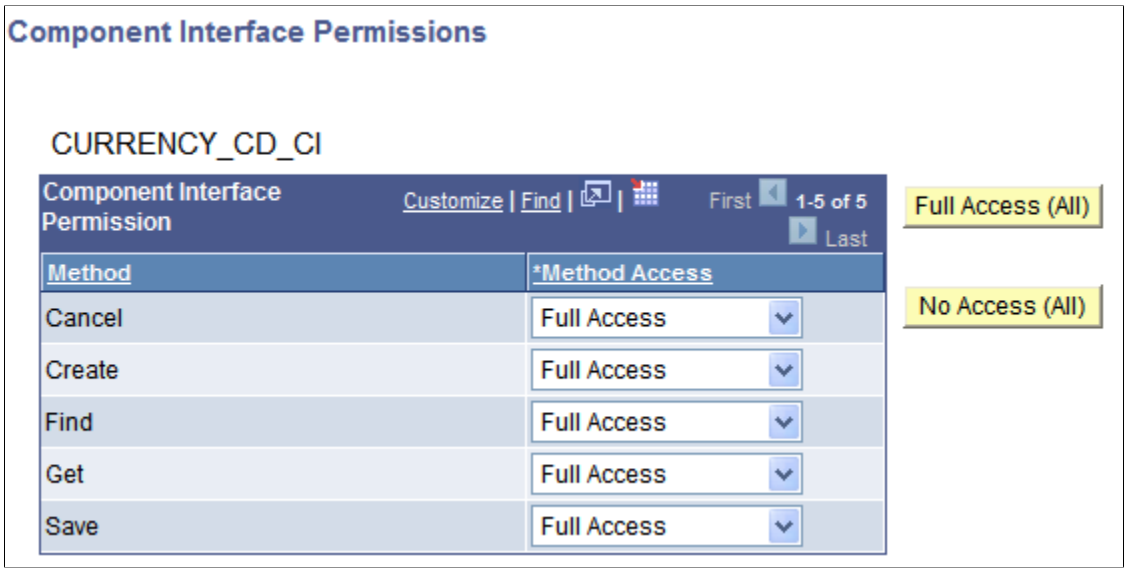
Setting Security Options

After creating a component interface, you must set security for it. Each individual method also needs to be provided security. Security for the component interface is provided through the PeopleSoft Internet

Architecture pages. Component interface permissions are set at the permission list level in PeopleSoft security.

Image: Component Interface Permissions page

This example illustrates the Component Interface Permissions page. Use the page to manage access to the methods of a component interface.



To set up component interface security:

1. Sign in to the PeopleSoft Pure Internet Architecture through the browser, and select PeopleTools > Security > Permissions & Roles > Permission Lists.
2. Select the permission list for which you want to set security.

   The Permission List component appears.
3. Access the Component Interfaces page.
4. Select the component interface for which you want to set security.

   To add another component interface to the list, click the Add button.
5. Click Edit.

   The Component Interface Permissions page appears, showing all of the methods (both standard and user-defined) in the component interface and their method of access.
6. Set the access permission for each method.

   Select *Full Access* or *No Access*. You must grant full access to at least one method to make the component interface available for testing and other online use.
7. Click OK when you are done.
8. Save the page.

Testing Component Interfaces

After setting the security for a component interface, you can test the contents and behavior using the component interface tester. You should test the component interface before using it in your external system. This proactive tool helps you discover problems with the underlying component or the component interface itself, including user-defined methods. When you are testing a component interface, real data from the database is used. Therefore, if you save the information that you change by calling the Save method, the information is changed in the database.

With the component interface tester, you can:

- Test the component interface in interactive mode.
- Retrieve history items.
- Test the standard, custom, and collection methods.

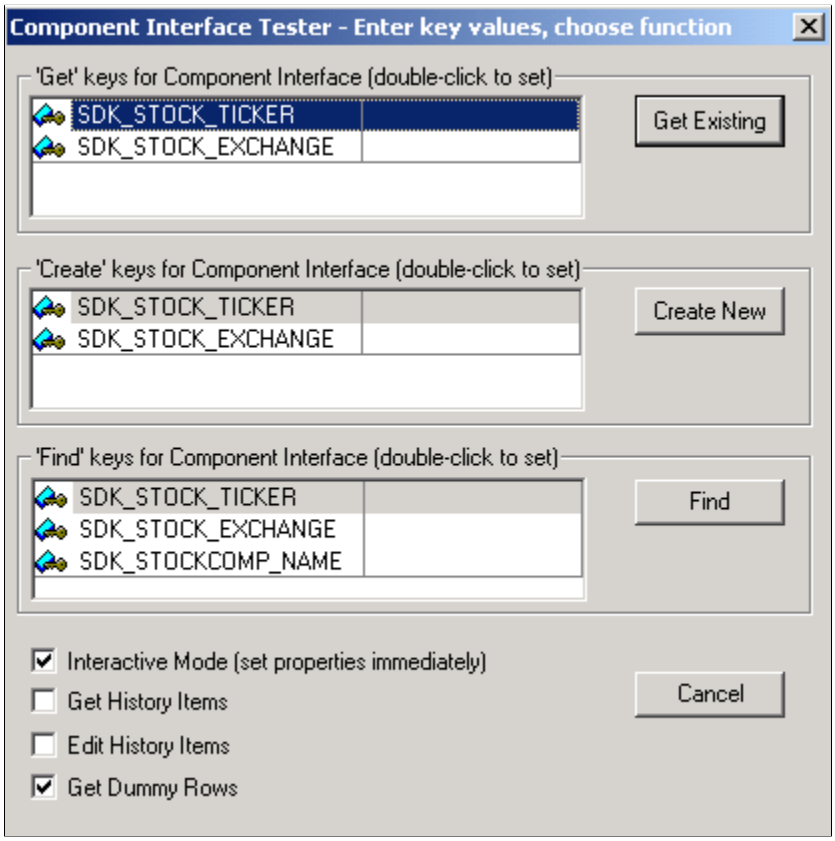
Searching Component Interfaces to Test

To test the component interface, you search for the component interface to test, and then you test it.

Access the Component Interface Tester search dialog box:

Image: Component Interface Tester – Enter key values, choose function dialog box

This example illustrates the fields and controls on the Component Interface Tester – Enter key values, choose function dialog box . You can find definitions for the fields and controls later on this page.



To search for a component interface to test:

1. Open the component interface in PeopleSoft Application Designer.
2. Select Tools > Test Component Interface from the PeopleSoft Application Designer menu.

   The Component Interface Tester search dialog box appears. This dialog box displays the keys (in the left-hand columns) for getting, creating, or finding an instance of the component interface. The right-hand columns provide a place for you to enter sample key values for testing.
3. Enter key values.
   a. Double-click the column to the right of any displayed keys.
   b. Enter the value in the right-hand column.

      The data that is used for the test corresponds to the key values that you enter here. In the preceding example, we have entered an employee ID of *6602*.

Interactive Mode	In interactive mode, any action request occurs immediately. Each property being set causes an immediate trip to the application server (or database server in two-tier mode). This differs from non-interactive mode, in which actions are often held and later sent in batches. For example, in non-interactive mode, if you set a property, the property is not validated until you perform the save. However, in interactive mode the property is validated immediately. This means that edit processing (and other processing, such as FieldChange PeopleCode) occurs for each set property. Whether you select this option depends on how you expect a particular component interface to be used and what you are currently testing. In a real production system, this parameter can significantly affect performance, but it makes little difference in the test component. In non-interactive mode, errors and properties are not updated until a method is run. By default, Interactive Mode is selected in the component interface tester.
Get History Items	Select to retrieves history data. This option applies to effective-dated fields only and is equivalent to running in either Update/Display or Update/Display All mode.
Edit History Items	Select to enable editing and saving of history data. This option applies to effective-dated fields only and is equivalent to running in either Update/Display or Correction mode.
Get Dummy Rows	Specify whether to get dummy rows. This option is selected as a default. The component processor provides dummy rows to enable quick data entry when the level you are accessing does not have any data. Because of this, an API that does not need this row finds it and exposes it to the user. The application that uses the API

now has to determine whether the row is a dummy row and accordingly decides to execute Item or InsertItem.

Setting the GetDummyRow to false enables the component interface processor to handle the counts accordingly. With this property set to false, users do not have to use item and InsertItem when adding new data at levels 1 to 3. Instead, they can comfortably always use InsertItem.

Get Existing

Clicking Get Existing is equivalent to opening a record in Update/Display or Correction mode online. It retrieves one instance from the database. After you click the Get Existing button, the Component Interface Tester dialog box appears.

Create New

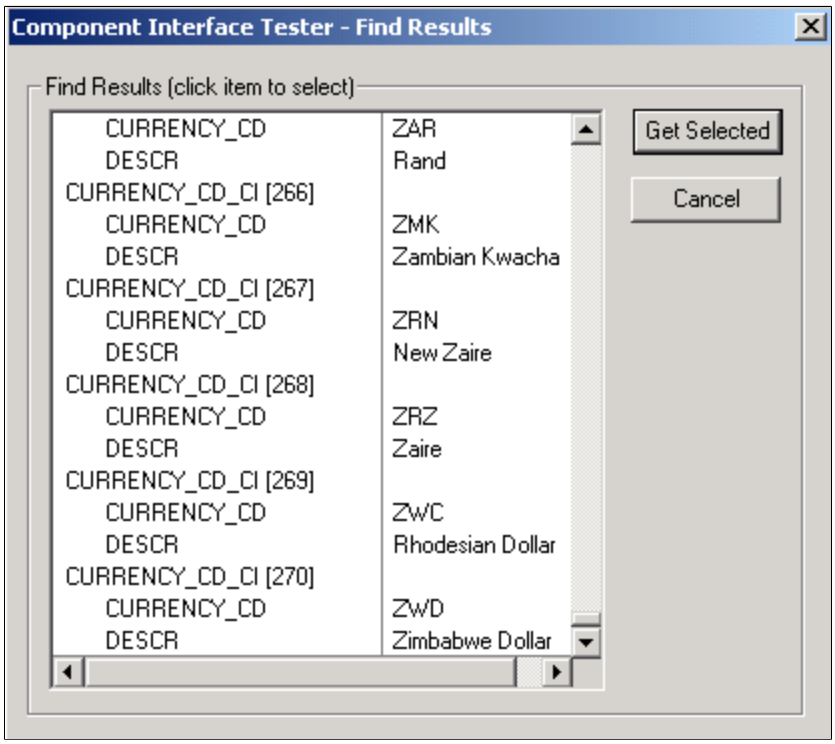
Clicking Create New is equivalent to creating a new row in Add mode online. If your component does not support the Create method, this button is disabled. After you click the Create New button, the Component Interface Tester dialog box appears.

Getting Existing Records by Using Partial Keys

If you want to retrieve a partial key, click the Find button on the Component Interface Tester page. The Find Results dialog box appears:

Image: Component Interface Tester — Find Results dialog box

This example illustrates the Component Interface Tester – Find Results dialog box. Use the page to get existing records by using partial keys.



You can choose the specific instance by selecting and clicking the Get Selected button. If you do not enter a partial key before clicking Find, all key values in the database are returned (subject to the maximum

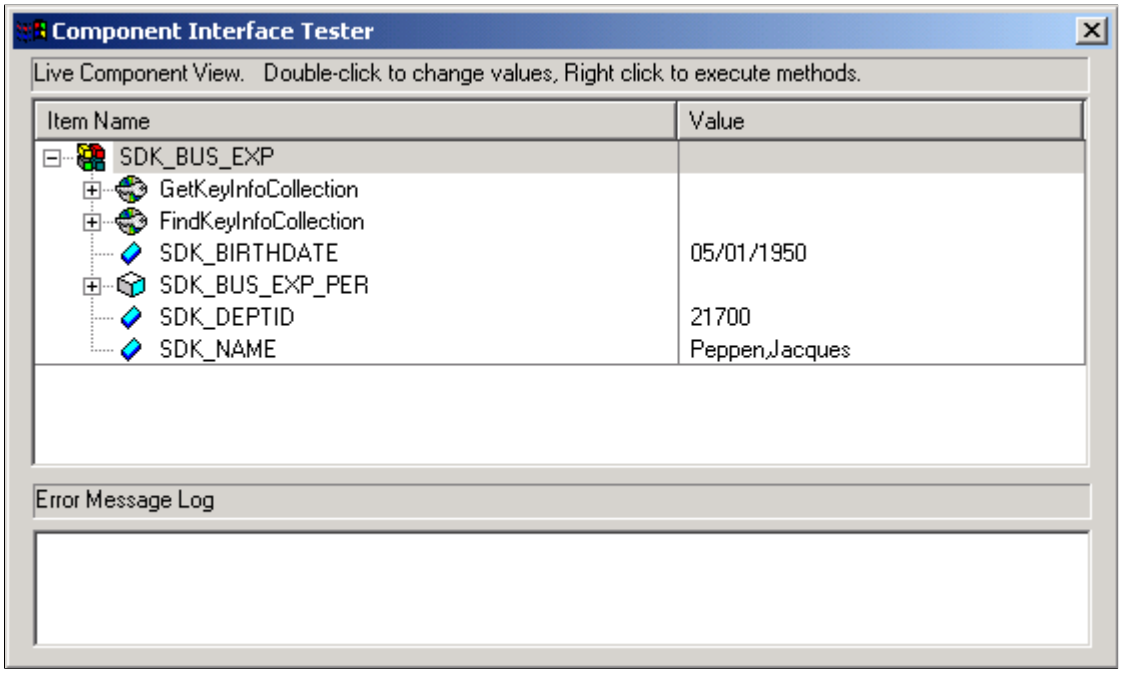
count of 300, just as when online). This is the same as calling the Find method through the component interface API, followed by selecting a value from the Find results, setting the Get key, and calling the Get method. After you click the Get Selected button, the Component Interface Tester dialog box appears.

Testing Component Interfaces

After you have searched for and retrieved the component interface, the Component Interface Tester dialog box appears.

Image: Component Interface Tester dialog box

This example illustrates the Component Interface Tester dialog box. Use the page to test component interface properties and methods.



Testing Component Interface Properties

From the Component Interface Tester dialog box, change the value of a property, double-click a value, and enter a new value. Some basic validation is done when you leave the field, which is equivalent to leaving a field using the Tab key in the online case. This validation includes system edit, FieldChange PeopleCode events, and FieldEdit PeopleCode events. Further validation can be done when the Save method is called (SaveEdit, SavePreChange, Workflow, and SavePostChange). If errors occur or warnings are encountered, messages are displayed in the Error Message Log area at the bottom of the window. The error message log displays the same text that would appear in the PSMessages collection of the Session object if you accessed the component through the Component Interface API.

Testing Component Interface Methods

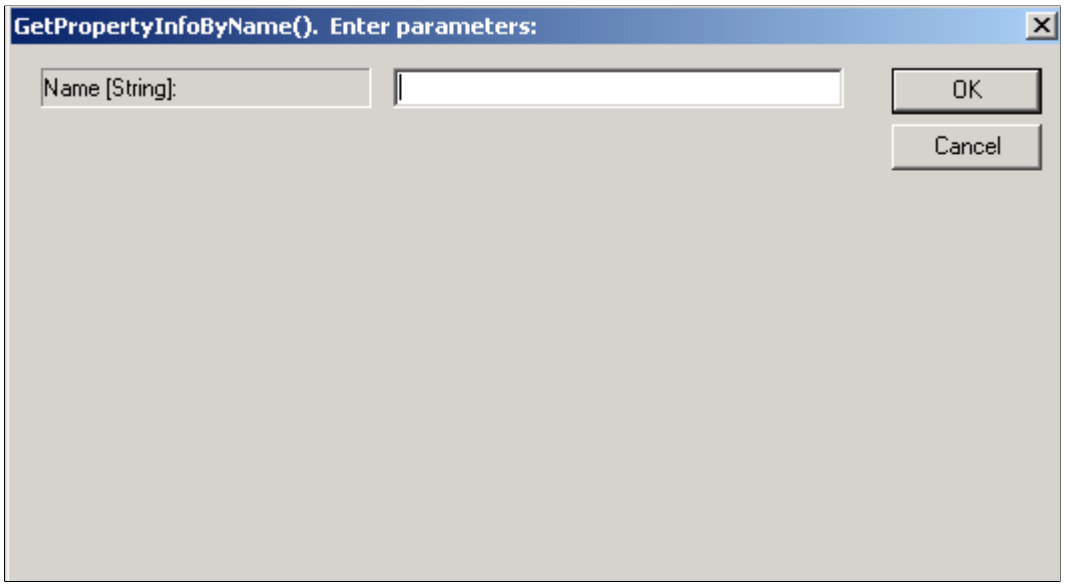
Test component interface methods by right-clicking the component interface name.

A menu appears that shows the Save and Cancel standard methods and any user-defined methods that exist for the component interface. The Find, Create, and Get standard methods are not valid for an instantiated component, and therefore are not shown.

If a component interface method requires one or more parameters, a dialog box in which you can enter the parameters appears. After the method is executed, the same dialog box appears again, displaying changes to the parameters that were caused by the method. The return value of the function appears in the title of the dialog box. If a component interface requires no parameters, you do not see the initial dialog box, but you do see the return value dialog box following the function call.

Image: Enter parameters dialog box

This example illustrates the Enter parameters dialog box. Use the page to enter required parameters for component interface methods.



Note: Because running a component interface method can result in a change to the component interface structure, PeopleSoft Application Designer always redraws the component interface tree in its collapsed form following a method call.

Testing Collection Methods

Test collection methods by right-clicking the collection name.

A menu appears that shows the standard collection methods. Select the collection method that you want to test for this component interface. After you select a collection method to test, the Enter parameters dialog box prompts you to enter an item number for the collection method that you are testing. The value that you enter for index [Number] is used to retrieve, insert, or delete an item, according to the following rules.

After you enter an index number, the result appears in the dialog box. If a return value is sent, it is displayed in the title bar. Otherwise, the message *No value* is displayed. Click OK or Cancel to dismiss the dialog box.

Collection Method Rules

This table describes the collection method rules:

Collection Method	*Purpose*
Item(index)	Returns the row at the specified index. Only the success or failure of this routine is of interest from inside the test component.
InsertItem(index)	Inserts a new row either before the index that you specify if the collection is effective-dated or following the index if it isn't effective-dated.
DeleteItem(index)	Deletes the row that is designated by the index number that you specified in the Enter parameters dialog box.
ItemByKeys(key1, key2, …)	Returns the row corresponding to the specified keys. Only the success or failure of this routine is of interest from inside the test component.
CurrentItem	This method returns the effective row in an effective-dated record. Only the success or failure of this routine is of interest from inside the test component.
GetEffectiveItem(DateString, SeqNum)	Returns the data row that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItem function, which returns the definition that is effective at this moment. This method works with effective-dated records only.
GetEffectiveItemNum(DateString, SeqNum)	Returns the item number inside the collection of the data row that would be effective for the specified date and sequence number. This is a more general case of the GetCurrentItemNum function, which returns the number of the definition that is effective at this moment. This method works with effective-dated records only.

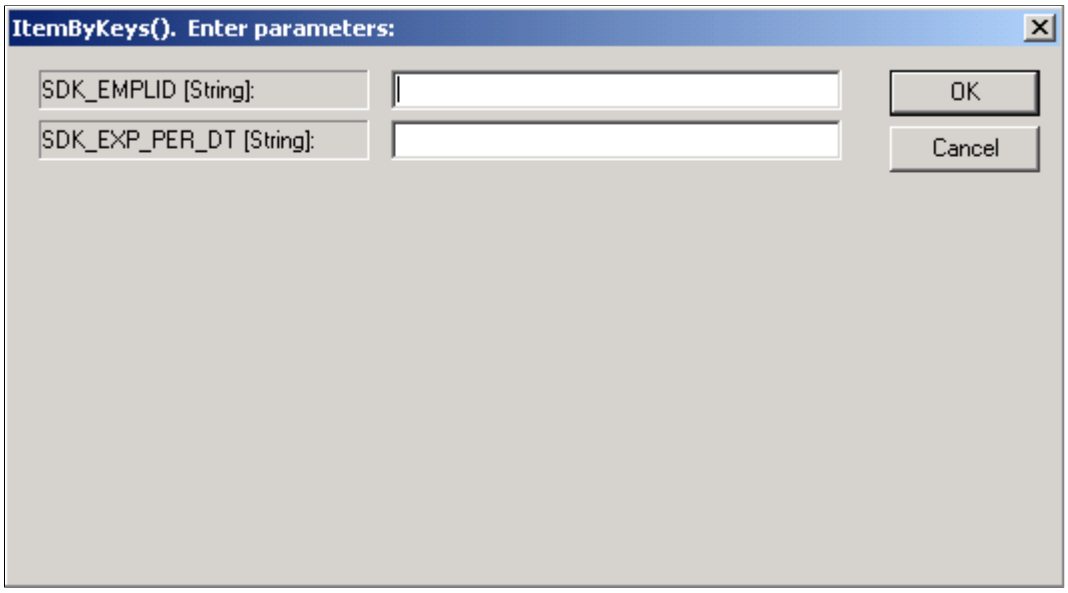
Note: Component interface classes contain information about collection methods.

Determining ItemByKeys Parameters

You can get the signature for the ItemByKeys method (or any other method) when testing a component interface. This is helpful for the ItemByKeys method, because its signature is different for each component interface.

Image: Enter parameters dialog box

This example illustrates the Enter parameters dialog box for the ItemByKeys method. Use the page to get the signature for any component interface.



To determine ItemByKeys parameters:

1. Open the definition.
2. Select Tools > Test Component Interface.
3. Find or get an appropriate populated component interface.
4. Navigate to the appropriate collection.
5. Right-click, and select ItemByKeys from the menu.

   A dialog box appears, showing the specific parameters and types and the order in which you should call ItemByKeys.

   In the preceding example, the keys for the SDK_BUS_EXP_PER ItemByKeys method are SDK_EMPID (String) and SDK_EXP_PER_DT (String).

Understanding Synchronization

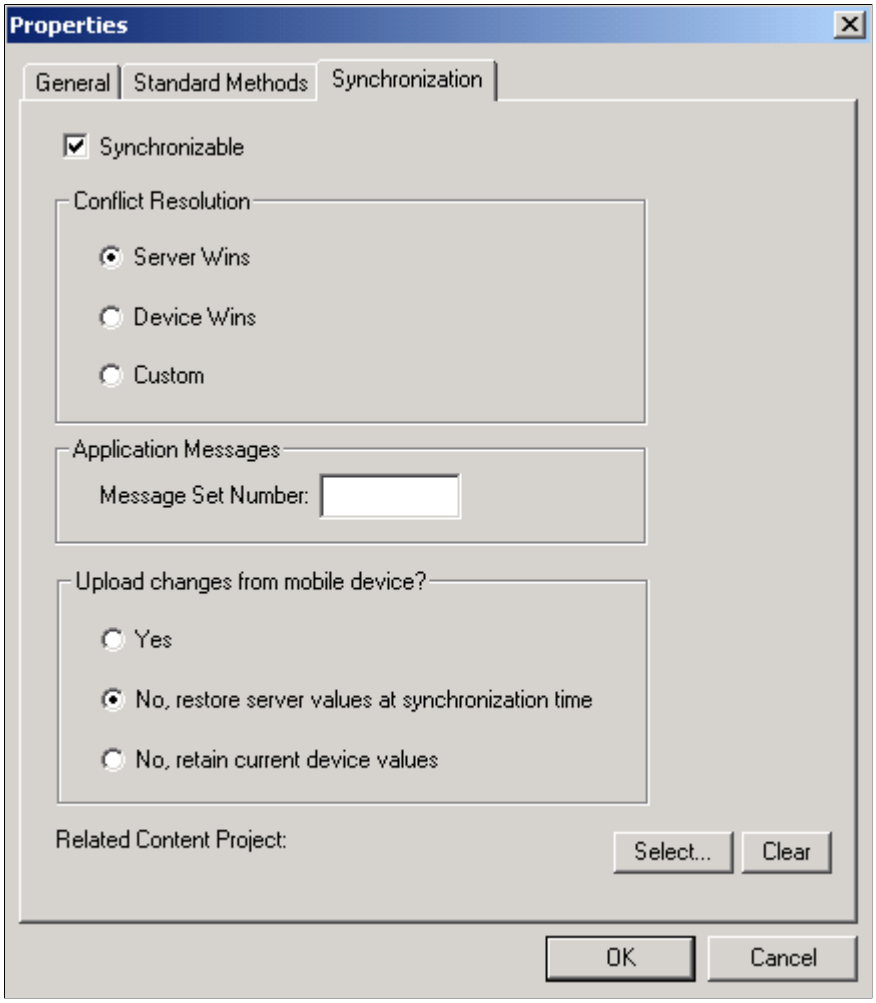
The Component Interface Properties Synchronization tab is used with PeopleSoft Mobile Agent.

Important! PeopleSoft Mobile Agent is a deprecated product. The Component Interface Properties Synchronization tab exists for backward compatibility only.

PeopleSoft Mobile Agent extends the functionality of PeopleSoft Pure Internet Architecture to disconnected mobile devices, enabling users to continue working with their PeopleSoft applications on a laptop computer or personal digital assistant (PDA) while disconnected from the internet or local network.

Image: Properties—Synchronization tab

This example illustrates the Properties—Synchronization tab.



Writing Component Interface Programs

The following chapters in this PeopleBook describe how to write component interface programs in several programming languages.

Also, the PeopleTools PeopleCode Reference contains a chapter that describes the component interface classes, including detailed instructions on the life cycle of a component interface and how to implement a component interface program in PeopleCode. You can use this information to help design your component interface program in other programming languages.

Related Links

"Understanding Component Interface Class" (PeopleTools 8.58: PeopleCode API Reference)

Understanding Runtime Considerations

In many ways, accessing a component interface is functionally equivalent to working with an online component. However, some important differences exist between component interfaces and components. This section describes how those differences affect interactive operation, functionality designed for graphical interfaces, client versus server operation, and several miscellaneous situations. These considerations, unless otherwise noted, apply to all the programming languages listed in this manual.

General Considerations

This section discusses general considerations for component interface programs.

WinMessage Unavailable

You cannot use the WinMessage API in a component that will be used to build a component interface. Use the MsgGet() function instead.

Email from a Component Interface

To use a component interface to send email, use the TriggerBusinessEvent PeopleCode event, not SendMail.

Related Display

Related display fields are not available for use in a component interface because they are not held in the buffer context that the component interface uses.

Row Inserts

If row insert functionality has been disabled for a page, you must take care when calling inserts against the corresponding component interface. Any PeopleCode associated with buttons used on the page to add rows will not be invoked by the component interface when an insert is done.

Note: If a component has logic that inserts rows on using the RowInsert event, the component interface cannot identify the change and locate the rows that were inserted by the application code. Generic interfaces such as Excel to Component Interfaces utility and the WSDLToCI will not function correctly when using this type of dynamic insert.

Custom Field Formats

Custom field formats that are defined dynamically via PeopleCode are not enforced by component interfaces, as they are evaluated by the page processor and not available to the component interface processor. Only the static formats defined in the PeopleSoft Application Designer will be applied.

Scope Conflicts

This section discusses scope conflicts for component interface programs.

Infinite Processing Loops

A component interface should not call itself in any of the PeopleCode included within its component definition, because this may result in an infinite loop of the component interface. A component interface also should not call itself from a user-defined method.

Multiple Instances of a Component Interface

C++ applications should not create multiple, simultaneous instances of the same component interface, either within a single procedure, or in both a parent and a child procedure because of potential conflicts.

Interactive Mode

This section discusses interactive mode considerations for component interface programs.

UNIX Server Performance

If you are using a component interface as part of a batch process in which thousands of rows are being inserted, running in interactive mode may reduce performance enough on some UNIX servers to produce a connection failure. Prevent this by setting the InteractiveMode property to False.

Hidden Edit Validation Errors

If the InteractiveMode property is set to True, and if a transaction sets a property to a value that isn't allowed in a prompt edit field, the edit field value is reset back to its original value. The error is logged in the PSMessages collection; however, the Save method runs without errors. Check the value of both the Save method and the collection ErrorPending property to discover all of the errors.

Chapter 4

Programming Component Interfaces Using PeopleCode

Understanding PeopleCode Behavior and Limitations

Note the behavior and limitations discussed in this section when you write PeopleCode for a component interface.

PeopleCode Event and Function Behavior

PeopleCode events and functions that relate exclusively to the graphical user interface and online processing cannot be used by component interfaces. These include:

- Search dialog processing.

  When you run a component interface, the SearchInit, SearchSave, and RowSelect events do not fire. This means that any PeopleCode associated with these events will not run. The first event to run is RowInit.

- Menu PeopleCode and pop-up menus.

  The ItemSelected and PrePopup PeopleCode events are not supported. In addition, the CheckMenuItem, DisableMenuItem, EnableMenuItem, HideMenuItem, and UncheckMenuItem functions are not available.

- Transfers between components, including modal transfers.

  The TransferPage, DoModalPageGroup, and IsModalPageGroup functions cannot be used.

- Dynamic tree controls.

  Functions related to this control, such as GetSelectedTreeNode, GetTreeNodeParent, GetTreeRecordName, RefreshTree, and TreeDetailInNode cannot be used.

- ActiveX controls.

  The PSControlInit and PSLostFocus events are not supported, and the GetControl function cannot be used.

- DoSave() and DoSaveNow().

  The DoSave() and DoSaveNow() PeopleCode functions are not supported. You should use the component interface Save() method and wrap the DoSave() and DoSaveNow() functions so that they do not execute when called from a component interface.

- Functions that are ignored in a component interface call.

Some PeopleCode functions are ignored if they are called through a component interface. These functions are:

- WinMessage
- CheckMenuItem
- DisableMenuItem
- EnableMenuItem
- HideMenuItem
- UncheckMenuItem
- SetCursorPos
- TransferPanel
- TransferPage
- DoModalComponent
- IsModalComponent
- DoModalPanelGroup
- IsModalPanelGroup
- GetSelectedTreeNode
- GetTreeNodeParent
- RefreshTree
- TreeDetailInNode
- GetControl
- DoSave
- DoSaveNow
- Gray
- Ungray

CopyRowset Language Considerations

In previous PeopleSoft releases, CopyRowset* functions for component interfaces were not sensitive to the language code on PSCAMA. Because of this, related language processing did not take place when language code on PSCAMA was different from the base language code. PeopleSoft now detects the language code in PSCAMA.

Limitations of Client-Only PeopleCode

Component interfaces can run on either the client or the server. By default, a component interface runs on the server. It runs on the client only if the code calling the component interface is running on a client machine.

Component interfaces must run either entirely on the server or entirely on the client. To ensure this runtime restriction, component interface references declared in PeopleCode must be declared as local, not global, variables.

Some built-in functions are always client-only; others are client-only under specific conditions.

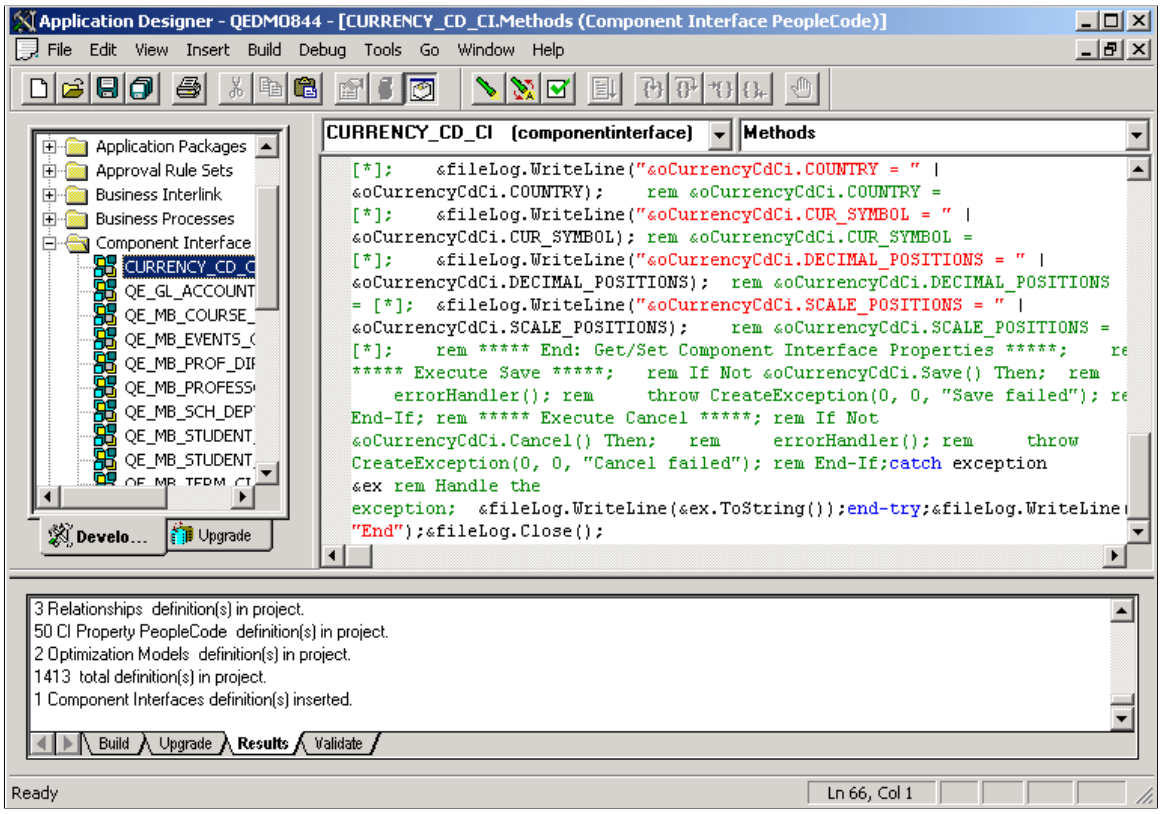
Some built-in functions behave differently when used in three-tier mode, as opposed to two-tier mode.

Generating PeopleCode Templates to Access Component Interfaces

To access a component interface using PeopleCode, PeopleSoft Application Designer generates a template in the form of boilerplate PeopleCode that you can adapt to your purposes. This section describes how to generate the template code.

Image: PeopleSoft Application Designer

This example illustrates PeopleCode generated by dragging and dropping a component interface.



To generate a PeopleCode template for a component interface:

1. Open the desired component interface definition in PeopleSoft Application Designer.

2. Insert the component interface into a project.

   a. Select Insert > Current Object into Project.

   b. Save the project.

3. Open the PeopleCode editor.

   You can associate component interface PeopleCode with a record, a component, a service operation handler, or an Application Engine program.

4. Select the component interface from the project workspace.

   Drag and drop the object from the project into the PeopleCode Editor.

5. Make any necessary changes to the PeopleCode in the PeopleCode Editor window.

   This is especially important on components that have multiple scrolls at the same level, as the automatic code generation may have difficulty determining the parent of the collection (scroll). Therefore, the template code should be inspected and corrected as needed.

Understanding PeopleCode Templates

The code shown in this section is a dynamically generated PeopleCode template that you can use as a starting point. Replace all default values or <*> notations with specific values or references to valid PeopleCode variables (replace this entire three-character string: <*>).

Note: The requirement to populate a non-create key is no longer a requirement to do the initial save.

PeopleCode runs only if you are connected. This means that you do not have to explicitly connect. Instead, connect to the existing session, using the %Session system variable.

You cannot connect to a different database through PeopleCode.

Set the PeopleSoft session error message mode. This property is used to determine how messages are output. This property takes either a numeric value or a constant. The default value is 1 (%PSMessages_CollectionOnly).

This property sets the value for the session. You can change modes during a session, for example, if you're starting a component interface. However, after you run the component interface, you should set the value back. Here is the list of modes that you can use:

Mode Value	***Description***
0	Return no messages.
1	Default. Log messages into the PSMessage collection.
2	Display a pop-up message or dialog box.
3	Log messages into the PSMessage collection and pop up a message dialog box.

PeopleCode Template Notes

Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)

Set the keys for the component interface. In this example SDK_EMPLID is the Get key.

The get() method retrieves data from the database, associated with the key values.

Get and print properties at level 0.

Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get collection at level 1 (SDK_BUS_EXP_PER).

Get and print properties at level 1.

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

```
&oSdkBusExpDtlCollection = &oSdkBusExpPer.SDK_BUS_EXP_DTL;
```

Get and print properties at level 2.

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

Related Links

"Session Class Properties" (PeopleTools 8.58: PeopleCode API Reference)

Chapter 5

Programming Component Interfaces in Java

Building APIs in Java

If you plan to access your component interface from a Java external application, you must create a component interface API. The APIs are in the form of *.java source code files, which should be compiled into Java classes.

To build the component interface bindings:

1. Open any component interface definition in PeopleSoft Application Designer.

   Use any component interface definition, because you can build APIs for all of them, regardless of which one is open.

2. Select Build > PeopleSoft APIs.

   The Build PeopleSoft API Bindings dialog box appears.

3. Select the Build check box in the Java Classes group box.

   For the target directory, enter the directory in which you want the Java class source files to be created.

4. Click OK to build the bindings that you selected.

   The files that constitute the bindings are built in the location that you specified. If the operation is successful, a Done message appears in the PeopleSoft Application Designer Build window.

5. Compile the APIs that you just generated.

   You could use one of these commands:

   - Example 1:

     ```
     cd %PS_HOME%\class\PeopleSoft\Generated\CompIntfc
     javac -classpath %PS_HOME%\class\psjoa.jar *.java
     ```

   - Example 2:

     ```
     cd c:\pt8\class\PeopleSoft\Generated\PeopleSoft
     javac -classpath %PS_HOME%\class\psjoa.jar *.java
     ```

Setting Up the Java Environment

When deploying component interfaces on a local client machine or web server with Java bindings, you must have:

- The third-party Java application.
- The application server and database.
- The Java Virtual Machine (JVM) supplied with Sun Microsystems Java Development Kit (JDK). The JDK may already be installed on your system. To verify that the JVM is installed, check the *%PS_HOME%\JRE* directory. If it is not installed, you can obtain download information at the Oracle web site at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.

To set up your client machine to access the component interface API using Java:

1. If it is not already installed on your system, install the Sun Microsystems JDK to enable the JVM.

   You can download the JDK to any location, for example *c:\bea\jkd<version>*.

2. Set the environment variable PATH to include the directory containing jvm.dll.

   For example, you might set it at *c:\bea\jkd<version>\jre\bin\client*; or, if the PeopleTools install is done locally, the path is *<PS_HOME>\jre\bin\client*.

3. Set the environment variable CLASSPATH to include:

   - The file psjoa.jar (typically *<PS_HOME>\class\psjoa.jar*).
   - The target directory selected during the Build API process (<PS_HOME>\class).

Note: In previous releases, sites using UNIX servers received the following error message when invoking a component interface through the PeopleSoft Java Object Adapter (PSJOA): *PSProperties not loaded from file.* To resolve this issue, copy the pstools.properties file to the component interface execution directory.

Generating Java Runtime Code Templates

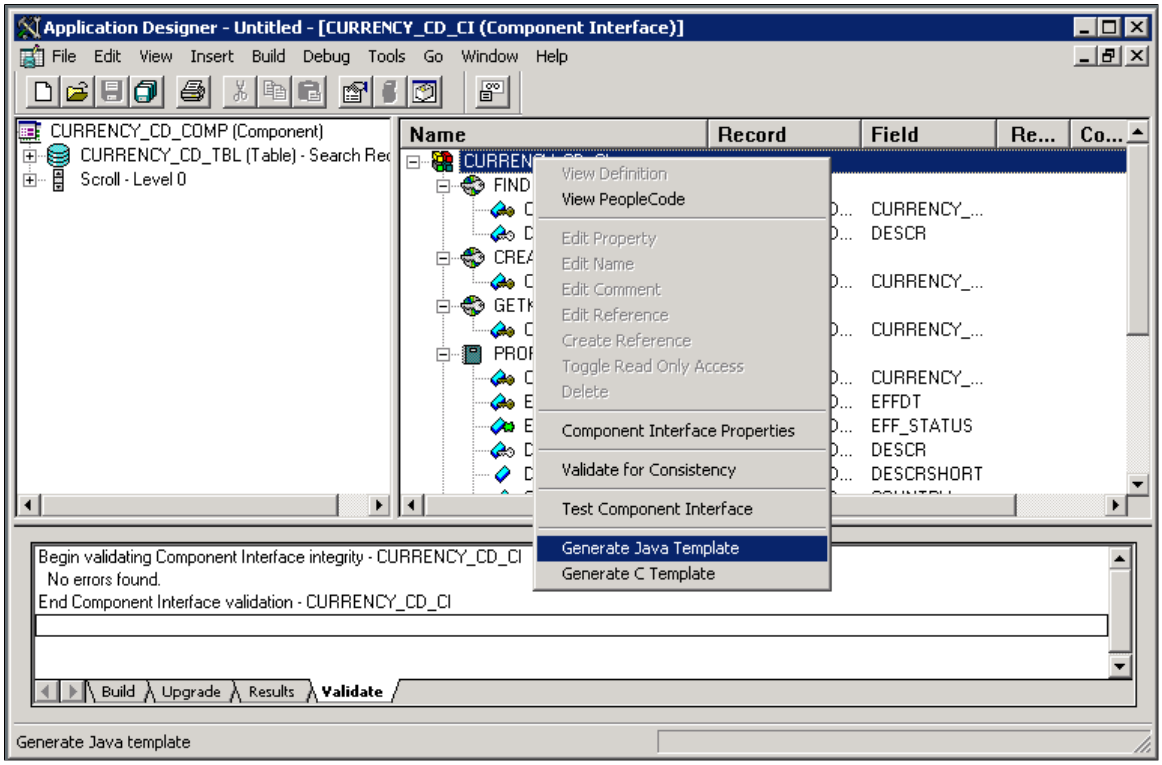
To access a component interface through external APIs using Java, PeopleSoft Application Designer generates a template in the form of boilerplate Java code that you can adapt to your purposes.

External Java APIs are located in the <PS_CFG_HOME>\ExtAPI_Java directory.

This section describes how to generate the template code.

Image: Generate Java Template Option

This example illustrates using PeopleSoft Application Designer to generate a Java runtime template.



To generate a Java template for a component interface:

1. Open a component interface definition in PeopleSoft Application Designer.
2. Right-click anywhere in the definition view to display the menu.
3. Select Generate Java Template.

   When the template is successfully generated, a message appears stating the name and location of the template file.

   Note: The template file is generated in the directory specified by the TEMP or TMP system environment variable on your client machine.

   ---

4. Edit the generated file and modify the source code to suit your needs.
5. Compile the source code to generate a *class* file.

   In the case of the example used in this manual, you could use this command:

   ```
   javac -classpath c:\temp;c:\pt8\class;c:\PT8\class\psjoa.jar SDK_BUS_EXP.java
   ```

Understanding the Java Template

You can use the Java template as a starting point for your Java program. This section contains a skeleton of the generated Java template for a component interface named SDK_BUS_EXP, which is part of the component interface SDK. The template has been edited for length.

Import all the required classes.

```
import java.io.*;
import psft.pt8.joa.*;
import PeopleSoft.Generated.CompIntfc.*;
public class SDK_BUS_EXP {
   public static ISession oSession;
   .....
   public static void main (String args[]) {
   try {
      //***** Set Connect Parameters *****
      String strServerName, strServerPort, strAppServerPath;
      String strUserID, strPassword;
      .....
      //Build Application Server Path
      strAppServerPath = strServerName + ":" + strServerPort;
```

Note: To enable Jolt failover and load balancing in the PeopleSoft Pure Internet Architecture, you can supply multiple application server domains for the strAppServerPath variable. Separate the domain names with a comma, and make sure that no spaces are included, for example, strAppServerPath = //APPSRVR1:8000,//APPSRVR2:9000

Create the PeopleSoft Session object to enable access to the PeopleSoft system.

The Session object controls the environment and enables you to do error handling for all APIs from a central location.

```
      //***** Create PeopleSoft Session Object *****
      oSession = API.createSession();
```

Connect to the application server by using the connect method of the Session object.

```
//***** Connect to the App Server *****
//if the Jolt Password is to be provided, switch to the the second
//version of the statement below
if (!oSession.connect(1, strAppServerPath, strUserID,
strPassword, null)) {
//if (!oSession.connectS(1, strAppServerPath, strUserID,
//strPassword, null, strJoltPwd)){
System.out.println("\nUnable to Connect to the Application Server.
Please verify it is running");
ErrorHandler();
return;
}
```

You define the domain connection password using the DomainConnectionPwd field in the Security section of the application server configuration file, configuration.properties. "Security Options" (PeopleTools 8.58: System and Server Administration)"Configuring Domain Connection Password" (PeopleTools 8.58: System and Server Administration)

Beginning in PeopleTools 8.53, using a domain connection password to connect to the application server is optional. However, if a domain connection password is specified, then the methods in the component interface APIs have to specify a value for it.

If the application server is configured to use a domain connection password other than the default value, use the connectS method, currently shown commented out in the previous Java template example, instead of the Connect method. The connectS method takes in all the same parameters as the Connect method, plus a domain Connection password as an additional parameter:

```
connectS(1, strAppServerPath, strUserID, strPassword, null, strJoltPwd);
```

Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)

```
  ISdkBusExp oSdkBusExp;
      String ciName;
      ciName = "SDK_BUS_EXP";
      oSdkBusExp = (ISdkBusExp) oSession.getCompIntfc(ciName);
      if (oSdkBusExp == null) {
         System.out.println("\nUnable to Get Component Interface " +
            ciName);
         ErrorHandler();
         return;
      }

      //***** Set the Component Interface Mode *****
      oSdkBusExp.setInteractiveMode(false);
      oSdkBusExp.setGetHistoryItems(true);
      oSdkBusExp.setEditHistoryItems(false);
```

Set the keys for the component interface. In this example, SDK_EMPLID is the Get key.

```
//***** Set Component Interface Get/Create Keys *****
      String strSdkEmplid;
      System.out.print("\nEnter SdkEmplid: ");
      strSdkEmplid = inData.readLine();
      oSdkBusExp.setSdkEmplid(strSdkEmplid);
```

The get() method retrieves data from the database, associated with the key values.

```
//***** Execute Get *****
      if (!oSdkBusExp.get()) {
         System.out.println("\nNo rows exist for the specified keys.
          \nFailed to get the Component Interface.");
         ErrorHandler();
         return;
      }
      .....
```

Get and print properties at level 0.

```
      System.out.println("oSdkBusExp.SdkName: " +
         oSdkBusExp.getSdkName());
      .....
```

Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get collection at level 1 (SDK_BUS_EXP_PER).

```
      ISdkBusExpSdkBusExpPerCollection oSdkBusExpPerCollection;
      ISdkBusExpSdkBusExpPer oSdkBusExpPer;
      oSdkBusExpPerCollection = oSdkBusExp.getSdkBusExpPer();
```

Get and print properties at level 1.

```
for (int i17 = 0;
         i17 < oSdkBusExpPerCollection.getCount(); i17++) {
         oSdkBusExpPer = oSdkBusExpPerCollection.item(i17);

         System.out.println("oSdkBusExpPer.SdkExpPerDt: " +
```



```
         oSdkBusExpPer.getSdkExpPerDt());
         .....
```

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

```
         ISdkBusExpSdkBusExpPerSdkBusExpDtlCollection
            oSdkBusExpDtlCollection;
         ISdkBusExpSdkBusExpPerSdkBusExpDtl oSdkBusExpDtl;
         oSdkBusExpDtlCollection = oSdkBusExpPer.getSdkBusExpDtl();
```

Get and print properties at level 2.

```
         for (int i211 = 0;
            i211 < oSdkBusExpDtlCollection.getCount(); i211++) {
            oSdkBusExpDtl = oSdkBusExpDtlCollection.item(i211);

            System.out.println("oSdkBusExpDtl.SdkChargeDt: " +
               oSdkBusExpDtl.getSdkChargeDt());
            .....
```

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

```
         }
      }
```

Disconnect from the Application server by using the disconnect method of the Session object. This method clears the buffers and releases the memory.

```
 //***** Disconnect from the App Server *****
      oSession.disconnect();
      return;
   }
   catch (Exception e) {
      e.printStackTrace();
      System.out.println("An error occurred: ");
      ErrorHandler();
   }
   }
}
```


Chapter 6

Programming Component Interfaces in C++

Building APIs for C++

If you plan to access your component interface from a C++ external application, you must create a component interface API. The APIs are in the form of C header files (*.h), which need to be included in the calling program.

To build the component interface bindings:

1. Open any component interface definition in PeopleSoft Application Designer.

   Use any component interface definition, because you can build APIs for all of them, regardless of which one is open.

2. Select Build, PeopleSoft APIs.

   The Build PeopleSoft API Bindings dialog box appears.

3. Select the Build check box in the C Header Files group box.

   For the target directory, enter the directory in which you want the C++ header file to be created, typically <PS_HOME>\bin\client\winX86.

4. Click OK to build the bindings that you selected.

   The peoplesoft_peoplesoft._i.h file that constitutes the bindings is built in the location that you specified. If the operation was successful, a *Done* message appears in the PeopleSoft Application Designer Build window.

Setting Up the C++ Environment

When deploying component interfaces on a local client machine with C++ bindings, you must have:

- The third-party C++ application.
- The Application server and database.
- The Java Virtual Machine (JVM) supplied with the Sun Microsystems Java Development Kit (JDK) found in the %PS_HOME%\JRE directory.
- Your compiler, configured for the C++ project.

Third-Party Applications

For applications written in C or C++, note that:

- The function names generated by the Build APIs process can be quite long. You may want to consider creating classes within your C++ code to mask this length throughout your program.
- When you create your installation for your C or C++ program, make sure that you include the setup of the path to the psapiadapter.dll.

Setting Up Client Machines to Access C++ APIs

To set up your client machine to access the component interface API using C++:

1. Install the PeopleSoft File Server.

   See *PeopleSoft Installation Guide, "Using the PeopleSoft Installer."*

2. Set the environment variable PS_HOME to point to the installed PeopleSoft PeopleTools directory (for example, c:\pt854).
3. Set the environment variable PATH to include the directory containing jvm.dll and the directory containing the PeopleTools client binaries.

   For example, %PS_HOME%\bin\client\winx86; or, if the PeopleTools installation is done locally, the path is <PS_HOME>\jre\bin\client.

4. Install the JVM supplied with the Sun Microsystems JDK. The JVM is located in the %PS_HOME% \JRE directory.
5. Set the environment variable CLASSPATH to include the psjoa.jar file (typically <PS_HOME>\class \psjoa.jar).

Configuring Compilers for C++ Projects

To configure a compiler for the C++ project:

Note: These instructions assume that you are using Microsoft Visual C++. If you use a different compiler, apply the equivalent settings for that product.

---

1. Create a new project in Microsoft Visual C++.
2. Select Tools > Options.
3. Select the Directories tab.
4. Click the New button in the Options dialog box.
5. Enter the path to the SDK include files, for example:

   ```
   C:\PT840\SDK\PSCOMPINTFC\SRC\C++\SAMPLES\INC
   ```

6. Click OK to save the options.
7. Open the Project Settings dialog box.
8. Select the C/C++ tab.
9. Select the General category.

10. Add PS_WIN32 to the preprocessor definitions.
11. Select the Link tab.
12. Select the Input category.
13. Specify the full path to psapiadapter.lib for the Object/library modules.

    This is typically <PS_HOME>\src\lib\psapiadapter.lib. Make sure that this is the only entry for psapiadapter.lib.
14. Click OK to save the settings.

Generating C++ Runtime Code Templates

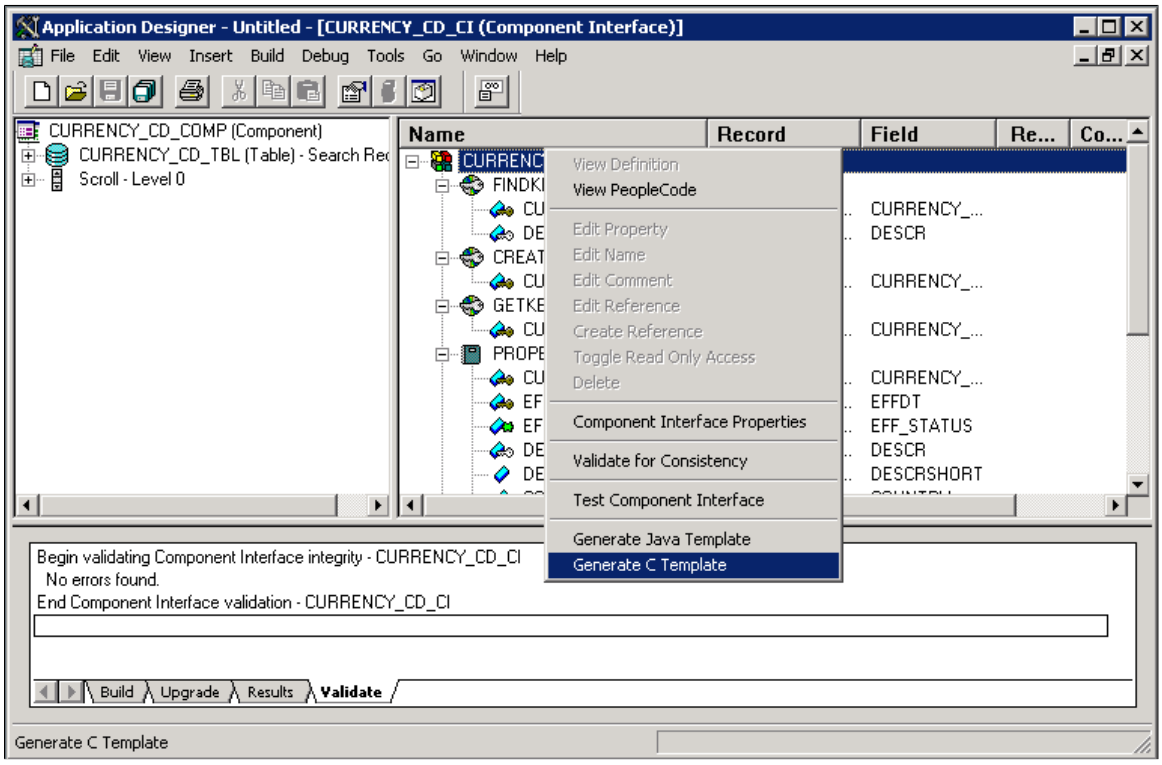
To access a component interface through external APIs using C++, PeopleSoft Application Designer generates a template in the form of boilerplate C++ code that you can adapt to your purposes.

External C++ APIs are located in the <PS_CFG_HOME>\ExtAPI_C directory.

This section describes how to generate the template code.

Image: Generate C Template Option

This example illustrates generating a C++ template in PeopleSoft Application Designer.



To generate a C++ template for a component interface:

1. Open a component interface definition in PeopleSoft Application Designer.
2. Right-click anywhere in the definition view to display the menu.

3. Select *Generate C Template.*

   When the template is successfully generated, a message appears stating the name and location of the template file.

   Note: The template file is generated in the directory specified by the TEMP or TMP system environment variable on your client machine.

4. Add the generated template file to the project.

   In Microsoft Visual C++:

   a. Open the project created earlier.

   b. Select Project > Add To Project > Files.

   c. Select the generated file.

   d. Click OK.

5. Edit the generated file and modify the source code to suit your needs.

6. Build the project to generate an executable (.exe) file.

Understanding the C++ Template

The C++ template can be used as a starting point for your C++ program. This section contains a skeleton of the generated C++ template for a component interface named SDK_BUS_EXP, which is part of the component interface SDK. The template has been edited for length.

Include all the required header files.

```
#ifdef PS_WIN32
#include "stdafx.h"
#endif

#include "cidef.h"
#include "apiadapterdef.h"
#include "PSApiAdapterInter.h"
#include "PSApiExternalLib.h"
#include "peoplesoft_peoplesoft_i.h"
#include <stdio.h>
#include <iostream.h>
#include <wchar.h>

HPSAPI_SESSION hSession;
TCHAR tmpValue[1024];

.....

void main(int argc, char* argv[])
{
   //***** Set Connect Parameters *****
   TCHAR strServerName[40], strServerPort[10], strAppServerPath[80];
   TCHAR strUserID[80], strPassword[80];
   .....

   //Build Application Server Path
   _stprintf(strAppServerPath, _T("%s:%s"), strServerName, strServerPort);
```


Note: To enable Jolt failover and load balancing in the PeopleSoft Internet Architecture, you can supply multiple application server domains for the strAppServerPath variable. Separate the domain names with a comma, and make sure that no spaces are included, for example, strAppServerPath = //APPSRVR1:8000,//APPSRVR2:9000.

Create the PeopleSoft Session object to enable access to the PeopleSoft system.

The Session object controls the environment and enables you to perform error handling for all APIs from a central location.

```
//***** Create PeopleSoft Session *****
PSAPIVARBLOB ExternalAuth;
memset(&ExternalAuth, 0, sizeof(PSAPIVARBLOB));
hSession = PSApiCreateSession();
if (!hSession)
{
   wprintf(L"\nUnable to Create Session\n");
   return;
}
```

Connect to the Application server by using the Session_Connect() function.

```
//***** Connect to the App Server *****
if (!Session_Connect(hSession, 1, strAppServerPath, strUserID,
  strPassword,ExternalAuth))
{
   wprintf(L"\nUnable to Connect to Application Server\n");
   ErrorHandler();
   return;
}
```

Get a reference to the component interface providing its name. (A runtime error occurs if the component interface does not exist.)

```
//***** Get Component Interface *****
HPSAPI_SDK_BUS_EXP hSdkBusExp;
TCHAR ciName[30];
_tcscpy(ciName, _T("SDK_BUS_EXP"));
hSdkBusExp = (HPSAPI_SDK_BUS_EXP) Session_GetCompIntfc(hSession,
   ciName);
if (!hSdkBusExp)
{
   wprintf(L"\nUnable to Get Component Interface %s\n", ciName);
   ErrorHandler();
   return;
}

//***** Set the Component Interface Mode *****
SdkBusExp_SetInteractiveMode(hSdkBusExp, false);
SdkBusExp_SetGetHistoryItems(hSdkBusExp, true);
SdkBusExp_SetEditHistoryItems(hSdkBusExp, false);
```

Set the keys for the component interface. In this example, SDK_EMPLID is the Get key.

```
//***** Set Component Interface Get/Create Keys *****
TCHAR strSdkEmplid[80];
wprintf(L"\nEnter SdkEmplid: ");
_getts(strSdkEmplid);
SdkBusExp_SetSdkEmplid(hSdkBusExp, strSdkEmplid);
```

The <CI_NAME>_Get() function retrieves data from the database associated with the key values.

```
//***** Execute Get *****
if (!SdkBusExp_Get(hSdkBusExp))
{
```



```
   wprintf(L"\nUnable to Get Component for the Search keys provided.\n");
   ErrorHandler();
   return;
}
```

Get and print properties at level 0.

```
wprintf(L"SdkBusExp.SdkName: %s\n",
   printProperty(SdkBusExp_GetSdkName(hSdkBusExp), tmpValue));
```

Similar code is generated for the properties SDK_BIRTHDATE and SDK_DEPTID.

Get collection at level 1 (SDK_BUS_EXP_PER).

```
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PERCOLLECTION
   hSdkBusExpSdkBusExpPerCollection;
HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER hSdkBusExpSdkBusExpPer;
hSdkBusExpSdkBusExpPerCollection =
    SdkBusExp_GetSdkBusExpPer(hSdkBusExp);
```

Get and print properties at level 1.

```
for (int i17 = 0; i17 < SdkBusExpSdkBusExpPerCollection_GetCount
                      (hSdkBusExpSdkBusExpPerCollection); i17++)
{
   hSdkBusExpSdkBusExpPer = SdkBusExpSdkBusExpPerCollection_Item
                       (hSdkBusExpSdkBusExpPerCollection, i17);
   wprintf(L"oSdkBusExpSdkBusExpPer.SdkExpPerDt: %s\n",
      printProperty
   (SdkBusExpSdkBusExpPer_GetSdkExpPerDt(hSdkBusExpSdkBusExpPer),
      tmpValue));
```

Similar code is generated for the properties SDK_EMPLID and SDK_BUS_EXP_SUM in the SDK_BUS_EXP_PER collection.

Get collection at level 2 (SDK_BUS_EXP_DTL).

```
   HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTLCOLLECTION
      hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection;
   HPSAPI_SDK_BUS_EXP_SDK_BUS_EXP_PER_SDK_BUS_EXP_DTL
      hSdkBusExpSdkBusExpPerSdkBusExpDtl;
   hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection =
   SdkBusExpSdkBusExpPer_GetSdkBusExpDtl(hSdkBusExpSdkBusExpPer);
```

Get and print properties at level 2.

```
   for (int i211 = 0; i211 <
       SdkBusExpSdkBusExpPerSdkBusExpDtlCollection_GetCount
      (hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection); i211++)
   {
      hSdkBusExpSdkBusExpPerSdkBusExpDtl =
         SdkBusExpSdkBusExpPerSdkBusExpDtlCollection_Item
         (hSdkBusExpSdkBusExpPerSdkBusExpDtlCollection, i211);

      wprintf(L"oSdkBusExpSdkBusExpPerSdkBusExpDtl.SdkChargeDt:
         %s\n", printProperty
         (SdkBusExpSdkBusExpPerSdkBusExpDtl_GetSdkChargeDt
        (hSdkBusExpSdkBusExpPerSdkBusExpDtl), tmpValue));
```

Similar code is generated for the properties SDK_EMPID, SDK_EXP_PER_DT, SDK_EXPENSE_CD, SDK_EXPENSE_AMT, SDK_CURRENCY_CD, SDK_BUS_PURPOSE, and SDK_DEPTID.

```
   }
}
```

Disconnect from the Application server by using the disconnect method of the Session object. This method clears the buffers and releases the memory.

```
   //***** Disconnect from the App Server *****
   Session_Disconnect(hSession);

   return;
}
```


Chapter 7

Using the Component Interface Software Development Kit

Understanding the Component Interface SDK

The PeopleSoft component interface SDK is installed with the PeopleTools installation. It provides resources to assist you in developing and testing component interface-based integration between PeopleSoft and third-party applications. The SDK contains sample definitions with data and source code. For easy identification, all of the definition names start with *SDK_*. The SDK is installed in the PeopleSoft home directory (PS_HOME) under *sdk*.

Note: The SDK definitions and associated data are for development purposes only and should not be used in a production environment.

Component Interface SDK Samples

Programming samples for the component interface SDK_BUS_EXP are part of the SDK. The samples are available in two languages—Java, and C++.

The component interface source code is located in the <PS_HOME>\SDK\PSCOMPINTFC directory.

Note: The source files mentioned in this section are located relative to the installed PeopleSoft home directory (PS_HOME).

Prerequisites for Using the Component Interface SDK

To call a PeopleSoft component interface, you must have:

- A working understanding of PeopleTools components.
- A working understanding of Java, or C++.
- The application server and database installed.
- The Java Virtual Machine (JVM) installed that is supplied with the Sun Microsystems Java Development Kit (JDK), found in the %PS_HOME%\JRE directory.

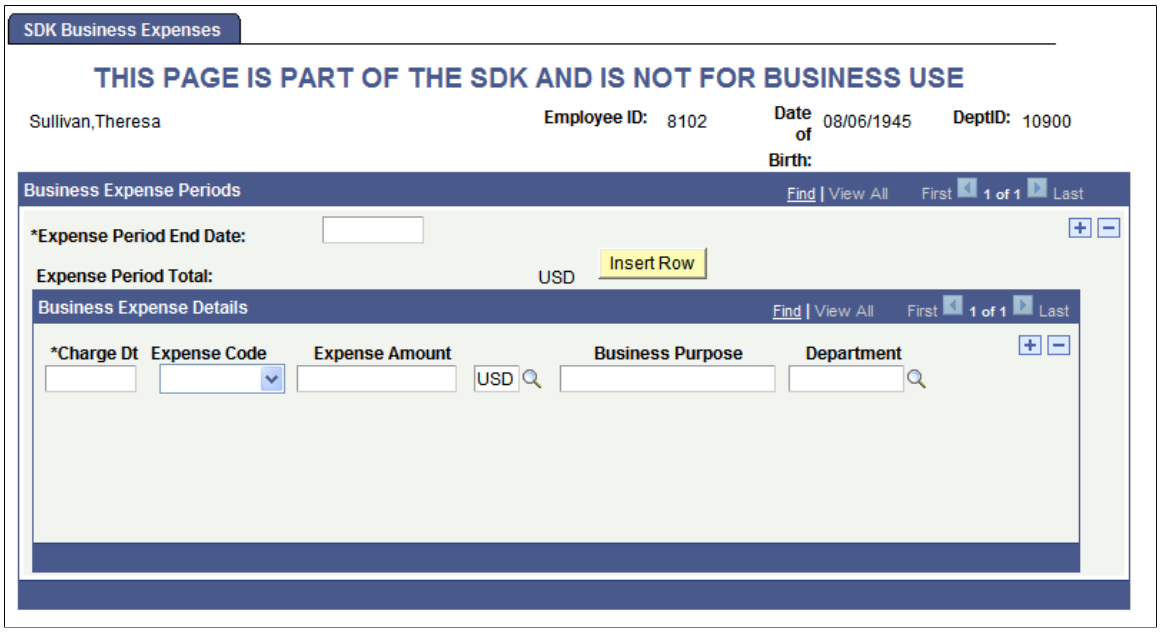
Using the SDK_BUS_EXPENSES Test Page

The SDK includes a component interface, called SDK_BUS_EXP, which is part of the sample development project and is delivered with the SDK. It is built on the component SDK_CI_SAMPLES, which contains the page SDK_BUS_EXP. The page exposes information about employee business expenses for external access.

Note: The component SDK_CI_SAMPLES is a sample and is not for business use.

Image: SDK_BUS_EXPENSES page

This example illustrates the SDK Business Expenses page.



To test the SDK_BUS_EXPENSES test page:

1. Provide access to the SDK_CI_SAMPLES component, using PeopleTools security.
2. Select PeopleTools SDK > PeopleTools SDK > Use > SDK CI Samples.
3. Search for and select an employee ID.

Testing the SDK_BUS_EXP Component Interface

To test the SDK_BUS_EXP component interface:

1. View the component interface definition through the PeopleSoft Application Designer.
2. Test the component interface definition, using the component interface tester.

Using the Component Interface SDK Sample in Java and C+ +

This section describes how to use the component interface SDK sample in Java and C++.

Understanding using the Component Interface SDK Samples in Java and C ++

The component interface sample programs for Java and for C++ are provided as part of the component interface SDK and follow the same sequence of options. The source files are located in <PS_HOME>\sdk \pscompintfc\src\<java or c++>\samples\sdk_bus_exp.

Building the Component Interface SDK Sample (Java)

The component interface sample program for Java is provided as part of the component interface SDK, located in <PS_HOME>\sdk\pscompintfc\src\java\samples\sdk_bus_exp.

The Java source code for the sample is in the following file: sdk_bus_exp.java

Before you run the sample, you must build the APIs and set up the Java environment.

To build the Java component interface sample:

1. Set your java classpath to include the external API classes you already built and the psoa.jar library delivered under <PS_HOME>\class\psjoa.jar
2. Compile the source using javac sdk_bus_exp.java

Building the Component Interface Sample (C++)

The component interface sample program for C/C++ is provided as part of the component interface SDK, located in <PS_HOME>\sdk\pscompintfc\src\cpp\samples\sdk_bus_exp.

The C++ source code for the sample is in the following file: sdk_bus_exp\sdk_bus_exp.cpp

Before you run the sample, you must build the APIs and set up the C++ environment. To build the C++ component interface sample:

1. Open the sdk_bus_exp workspace in the Microsoft Visual C++ editor.
2. Build the project by selecting Build, Rebuild All.

Running the Component Interface SDK Sample in Java and C++

To run the compiled Java or C++ component interface sample:

1. In a DOS window, change directories to the location of the sdk_bus_exp directory.

   After you launch the executable sdk_bus_exp, the system prompts you for parameters one at a time.

2. At each prompt, enter the appropriate value and press Enter.

   Select option 1 to sign in. You are then prompted to provide the connect information.

If the connect succeeds, a menu appears where you can perform Get or Find functions.

3. Get details for an employee.

   Select option 1 to get details for an employee. You are then prompted with the different update modes and the employee ID for which you want to display information. Enter the employee ID 8001 and press Enter. This displays the level 0 data and the options that you can perform.

4. Select a business expense period at collection level 1.

   Select option 8, *Item,* to select a business expense period. Selecting this option displays a list of available business expense periods for the selected employee.

   Select the expense period that you want to work with.

5. Select a business expense detail item at collection level 2..

   Select option 18, *Item,* to select a business expense detail within the selected business expense period. Selecting this option displays a list of available business expense details within the selected business expense periods.

Interpreting the Code for the Component Interface SDK Sample (Java)

The following discussion refers to the Java sample program, sdk_bus_exp.java. (The code has been edited for length.) It explains the runtime options shown above.

1. Import all the required classes:

   The code example shows how to import the required classes:

```
package sdk_bus_exp;
import java.io.*;
import java.util.*;
import psft.pt8.joa.*;
import PeopleSoft.Generated.CompIntfc.*;
public class sdk_bus_exp {
.....
```

2. Declare all the required objects.

   Only one active period and one active detail record are possible at any time. Users are prompted to select the needed values if they are not active.

Collection Object	*Level*	*Item Object for Collection*
oSdkBusExpCollection	Root (SDK_BUS_EXP)	oSdkBusExp
oSdkBusExpPerCollection	Level 1 (SDK_BUS_EXP_PER)	oSdkBusExpPer
oSdkBusExpDtlCollection	Level 2 (SDK_BUS_EXP_PER_DTL)	oSdkBusExpDtl

   In addition, the CompIntfPropInfoCollection object is used to access the structure of a component interface. It is not specific to a component interface.

3. Declare the PeopleSoft session object.

4. Connect to the application server.
5. Instantiate the component interface.
6. Perform a Get or Create to access the component interface.

   You must provide the keys to access the record that you want to modify.
7. Use the appropriate methods to access the component interface properties.

   There are standard methods and user-defined methods defined for the session, the component interface, and the component interface collections.

   The executeMethod function is used to invoke a method specified as a function parameter (nMethodIn).

The component interface Java SDK sample has 25 options:

SDK Option	***Where Executed***
1 through 5	On the component interface.
6 through 15	SDK_BUS_EXP_PER collection.
16 through 25	SDK_BUS_EXP_DTL collection

Options 1 through 4 and options 6 through 25 are similar in behavior to those described in the product documentation for PeopleCode API Reference for a component Interface and its collections.

Option 5, InsertBusExpDtlDefaults, is the user-defined method of the SDK_BUS_EXP component interface. This method is defined in PeopleCode inside the component interface definition.

The logic used in the corresponding options of these collections is identical.

This is the main method. It performs such functions as starting the session, getting the component interface, and disconnecting:

```
public static final void main(String[] args)System.out.println("  ");
System.out.println("\t 1) Sign In  ");
System.out.println("\t q) Quit ");
System.out.println("  ");
System.out.print("Command to execute (1, q) [1]: ");
charTemp = readCharacter();
switch (charTemp) {case 'q':case 'Q':.....
disconnectFromAppServer();
return;
default:
getConnectParameters();
if (connectToAppServer()) {
oSdkBusExp = (ISdkBusExp) oSession.getCompIntfc(m_strCIName);
while (getKeyType()) {
methodInt = selectMethod();
while (methodInt != 0) {
executeMethod(methodInt);
if (methodInt == 2) {
methodInt = 0;
} else {
methodInt = selectMethod();
.....
```


Related Links

"Understanding Component Interface Class" (PeopleTools 8.58: PeopleCode API Reference)

Interpreting the Code for the Component Interface SDK Sample (C++)

The following listings of code are taken from the C++ sample program, sdk_bus_exp.cpp. (The code has been edited for length.)

1. Include all the headers.

```
#ifdef PS_WIN32
#include "stdafx.h"
#endif
#include "cidef.h"
#include "apiadapterdef.h"
#include "PSApiExternalLib.h"
#include "PSApiAdapterInter.h"
#include "PeopleSoft_PeopleSoft_i.h"
#include <stdio.h> #include <stdlib.h>
#include <iostream.h>
#include <wchar.h>
```

2. Declare the PeopleSoft session handle.

```
HPSAPI_SESSION hSession;
```

3. Declare all the required objects. Only one active period and one active detail record are possible at any time.

Collection Object	*Level*	*Item Object for Collection*
hSdkBusExpCollection	Root (SDK_BUS_EXP)	hSdkBusExp
hSdkBusExpPerCollection	Level 1 (SDK_BUS_EXP_PER)	hSdkBusExpPer
hSdkBusExpDtlCollection	Level 2 (SDK_BUS_EXP_PER_DTL)	hSdkBusExpDtl

Collection ObjectLevelItem Object for CollectionhSdkBusExpCollectionRoot (SDK_BUS_EXP)hSdkBusExphSdkBusExpPerCollectionLevel 1 (SDK_BUS_EXP_PER)hSdkBusExpPerhSdkBusExpDtlCollection Level 2 (SDK_BUS_EXP_PER_DTL)hSdkBusExpDtl

The function executeMethod is used to launch the appropriate method depending upon the user input (nMethodIn).

The component interface C++ SDK sample has 25 options:

SDK Option1 through 5 SDK_BUS_EXP_PER collection.6 through 25SDK_BUS_EXP_DTL collection

SDK Option	*Where Executed*
1 through 5	On the component interface.
6 through 15	SDK_BUS_EXP_PER collection
16 through 25	SDK_BUS_EXP_DTL collection

Options 1 through 4 and options 6 through 25 are similar in behavior to those described in the product documentation for PeopleCode API Reference for a component Interface and its collections.

Option 5, InsertBusExpDtlDefaults, is the user-defined method of the SDK_BUS_EXP component interface. This method is defined in PeopleCode inside the component interface definition.

The logic used in the corresponding options of these collections is identical.

Related Links

"Understanding Component Interface Class" (PeopleTools 8.58: PeopleCode API Reference)

Chapter 8

Using the Excel-to-Component Interface Utility

Understanding the Excel-to-Component Interface Utility

Use the Excel to Component Interface utility and component interfaces to upload data from Microsoft Excel into PeopleSoft databases. Each source workbook contains both worksheets and Excel Visual Basic code modules that execute business logic for each transaction.

Use the Microsoft Excel workbooks as a template to create worksheets that are specific to the business logic that you need to use when you are uploading data to the PeopleSoft system. You can copy the data input sheet to other workbooks for distribution without copying the code modules.

The code formats spreadsheet data into a PeopleSoft readable Document Object Model (DOM) structure, and submits it to the PeopleSoft database. Next a PeopleCode program parses the DOM structure and uses the component interface to create entries in the PeopleSoft database, validating the data submitted against the business logic that is built into the PeopleSoft component. Because the component interface is a wrapper around the component, all logic applied during data entry is applied when you are loading data through this tool.

The component interface executes all the necessary PeopleCode events and the field-level edits. Based upon results from saving the component interface, another DOM is created in the PeopleCode that returns success, warnings, errors, or a combination of the three to the Microsoft Excel document. Records in error can be corrected and resubmitted.

Prerequisites for Using the Excel to CI Utility

To use the Excel to CI utility you must have the following software installed.

Check the My Oracle Support web site for the currently certified versions of software supported.

- Microsoft Excel.
- Microsoft Visual Basic 6.0 SP5: Run-Time Redistribution Pack.

  You can download this software from the Microsoft website at http://www.microsoft.com/downloads/Search.aspx?displaylang=en.
- Microsoft Core XML Services (MSXML) 6.0 or higher.

  You can download this software from the Microsoft website at http://www.microsoft.com/downloads/Search.aspx?displaylang=en.

Understanding Building Component Interfaces for the Excel to Component Interface Utility

To use the Excel to Component Interface utility effectively, you must have a complete understanding of the component that you are using and the component interface that is built around it. In addition, you should know what data needs to be entered and which fields on the component need to be exposed as component interface properties. Fields that are not relevant for data input should not be exposed on the component interface. This reduces processing time when you are loading data, as well as saving time when you are building the template because no need to delete unnecessary properties on the template will exist.

Some component interface structures will need to be modified before they can be used to load data through the utility. Components that have logic to insert multiple rows in child collections, and then require more values to be set on those collections, will need modification to the component to work with the Excel to Component Interface utility. Change the component so that the logic to insert and partially populate these rows does not happen by default through the component interface.

%CompIntfc and %CompIntfcName can be used so that this logic does not fire either from any component interface or from the component interface that you created for use with the Excel to Component Interface utility.

Additionally, components that have no keys at level 0, but rely on logic at level 0 to load the level 1 collection, cannot be loaded by using the Excel to Component Interface utility.

Component interfaces that rely on CommitWork to save the data cannot be used in the Excel to Component Interface utility.

Prompt and translate table values are validated when data is saved and submitted to the database through the Excel to Component Interface utility. This is different from the behavior on the page when prompts and translates are validated interactively. Some components may use prompts that are dynamically populated. For those situations, you must know what the valid values for the prompt will be.

Note: Remember that any changes made to the structure of a component interface will also need to be reflected in the template. Always ensure that the component interface and the template in the Excel to Component Interface utility are in sync. Structural changes made in only the component interface will cause an error in the Excel to Component Interface utility when data is submitted to the database.

Testing Component Interfaces

Before using the Excel to Component Interface utility run the component interface through the component interface tester in three-tier mode. Testing the component interface enables you to troubleshoot any problems before running the component interface through the utility. If the component interface does not work in the tester, it will not work in the Excel to Component Interface utility either. The component interface tester is located on the Tools menu in PeopleSoft Application Designer.

See Testing Component Interfaces.

Performance Expectations

The performance of a component interface depends entirely upon the underlying component. If the component has a complex user interface with many pages and scrolls, the component interface generally will have a slower processing time. The best performance times are found with small and medium-complexity component interfaces.

PeopleCode Behavior and Limitations

Certain PeopleCode functions and events that are specific to the user interface do not execute through the component interface. You will need to modify PeopleCode for the component, pages, and records when you build the component interface for the component.

PeopleCode events and functions that relate exclusively to the page interface and online processing cannot be used by component interfaces. These include:

- Search dialog processing.
- Menu PeopleCode and pop-up menus.
- Transfers between components, including modal transfers.
- Dynamic tree controls.
- ActiveX controls.
- DoSave and DoSaveNow.
- Functions that are ignored in a component interface call.

Related Links

Understanding PeopleCode Behavior and Limitations

"Understanding Component Interface Class" (PeopleTools 8.58: PeopleCode API Reference)

Default Properties

When you create a new component interface in PeopleSoft Application Designer, the system can create default properties for all the fields exposed on the component interface that meet certain criteria.

When you are creating a new component interface, the following requirements must be met to qualify as a default property.

The fields should be of the following types:

- Character
- Long character
- Number
- Signed number
- Date

- Time
- Datetime

The field should be one of the following page control types and must be exposed on the page:

- Edit box
- Drop-down list box
- Check box
- Radio button

Related Links

Creating Component Interface Definitions

Running the Excel to Component Interface Utility

This section discusses steps to run the Excel to Component Interface utility.

Before starting to complete these tasks, review the prerequisites for using the Excel-to-CI utility.

Related Links

Understanding the Excel-to-Component Interface Utility

Granting Access to the WEBLIB_SOAPTOCI iScript

To use the Excel to Component Interface utility, you must grant access to the iScript WEBLIB_SOAPTOCI in the permission list of the user who is building the template.

Related Links

"Understanding Permission Lists" (PeopleTools 8.58: Security Administration)

"Managing Permission Lists" (PeopleTools 8.58: Security Administration)

Enabling the Developer Menu in Microsoft Excel 2007 and Later Versions

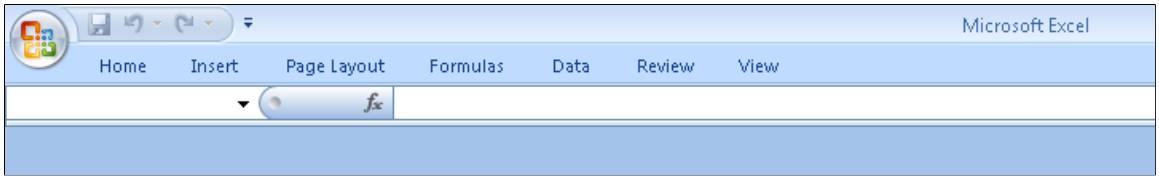
The Developer menu in Microsoft Excel contains options to work with Microsoft Visual Basic, macros, sheet properties, and so on.

In Microsoft Excel 2007 and later versions the Developer menu is not automatically enabled and does not appear on the menu ribbon in the default view of the Excel workspace. In the other versions of Microsoft Excel supported for use with the Excel to Component Interface utility, the Developer menu appears by default.

The following example shows the menu ribbon that appears in the default Microsoft Excel 2007 workspace view:

Image: Microsoft Excel 2007

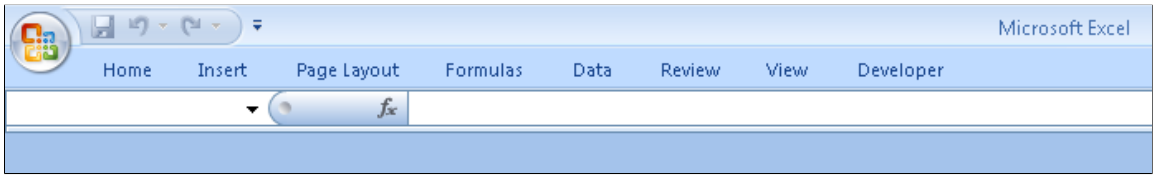
This example illustrates the menu ribbon that appears in the default Microsoft Excel 2007 workspace.



To use the Excel to Component Interface utility, you need access to some of the features accessed through via the Developer menu, and therefore you must enable the menu. The following example shows Microsoft Excel 2007 workspace with the Developer menu enabled on the menu ribbon:

Image: Microsoft Excel 2007

This example illustrates the Developer menu enabled on the Microsoft Excel 2007 menu ribbon.



Once enabled, the Developer menu appears on the far right on the menu ribbon.

To enable the Developer menu in Microsoft Excel 2007 and later versions:

1. Launch Microsoft Excel 2007 or your later version.
2. In the upper left corner of the workspace, click the circular Microsoft Office icon.

   The Recent Documents menu appears.
3. Click the Excel Options button at the bottom of the menu.

   The Excel Options page appears.
4. In the Top Options For Working with Excel section, check the Show Developer Tab in the Ribbon option.
5. Click the OK button.

The Microsoft Excel 2007 workspace appears and the Developer menu appears on the menu ribbon.

Enabling Macros in Microsoft Excel

The Excel to Component Interface utility relies on macros; therefore, you must enable macros in Microsoft Excel for the utility to work. When a Microsoft Excel spreadsheet is opened, the system displays a dialog box asking you to select whether to enable macros on the spreadsheet. Always select Enable Macros so that the macros delivered with the Excel to Component Interface utility can function.

To ensure that the macros are available to run, you must set the security level in Microsoft Excel to allow macros to open.

To enable macros in Microsoft Excel:

1. Open the Excel to Component Interface utility.
2. From the Excel menu, select Tools > Macros > Security.
3. Select either Medium or Low to enable the macros.
4. Select OK.

Starting the Excel to Component Interface Utility

The Excel to CI Utility spreadsheet is located in the PS_HOME/excel directory. The file name is ExcelToCI2007.xlsm.

Converting Excel to Component Interface Utility Templates to the Current Excel Version

You can use customized Excel-to-CI templates based on versions of Microsoft Excel released previous to Excel 2007. To preserve the macros embedded in your customized Excel-to-CI templates, you must convert the templates to Excel 2007 format.

Microsoft Excel 2007 files have the extension *.xlsm*.

To convert an Excel-to-CI template to Microsoft Excel 2007 format:

1. Open a template in Excel 2007.
2. Click the Microsoft Office Button and choose Save As.
3. Click Excel Macro-enabled Workbook.

   A Save As dialog box appears.
4. Choose a save location and enter a name for the workbook.
5. The workbook name must have the extension *xlsm,* such as *myworkbook.xlsm*
6. Click the Save button.

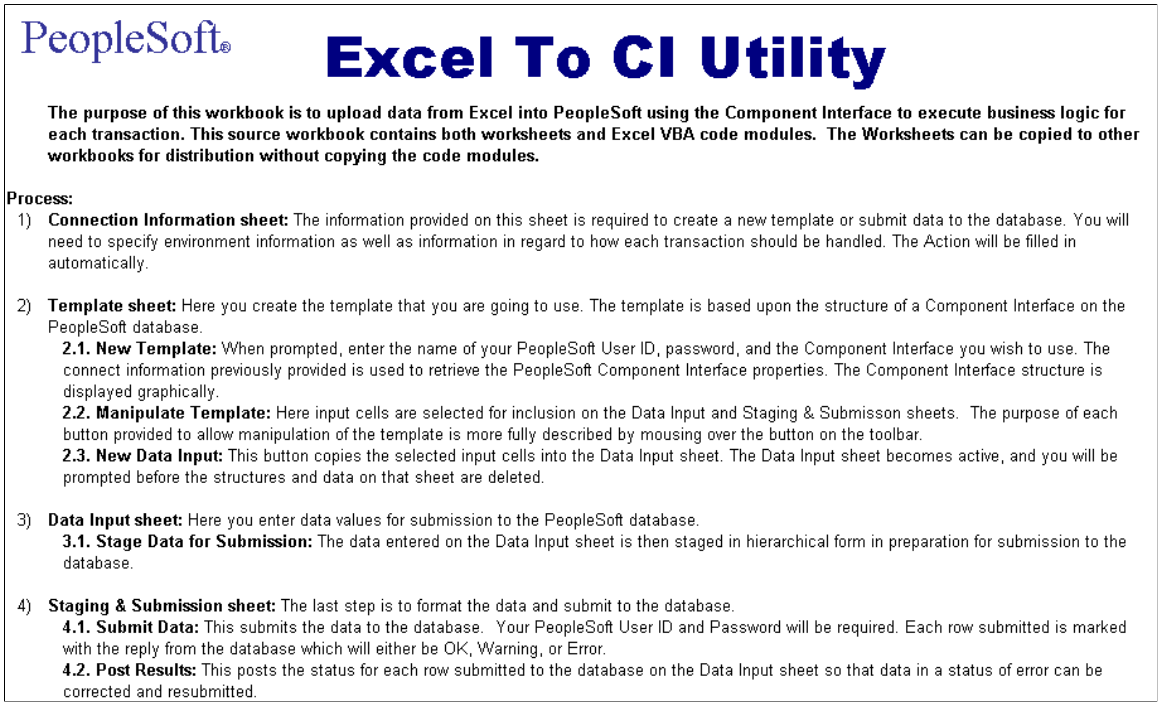
Viewing the Excel to Component Interface Coversheet

The coversheet of the Excel to Component Interface utility workbook gives a brief overview of the process flow and functionality of the tool.

Access the Coversheet tab in ExcelToCI2007.xlsm:

Image: Excel to CI Utility coversheet

This example illustrates the coversheet for the Excel to Component Interface utility.



PeopleSoft®

Excel To CI Utility

The purpose of this workbook is to upload data from Excel into PeopleSoft using the Component Interface to execute business logic for each transaction. This source workbook contains both worksheets and Excel VBA code modules. The Worksheets can be copied to other workbooks for distribution without copying the code modules.

Process:

1) **Connection Information sheet:** The information provided on this sheet is required to create a new template or submit data to the database. You will need to specify environment information as well as information in regard to how each transaction should be handled. The Action will be filled in automatically.

2) **Template sheet:** Here you create the template that you are going to use. The template is based upon the structure of a Component Interface on the PeopleSoft database.
   2.1. New Template: When prompted, enter the name of your PeopleSoft User ID, password, and the Component Interface you wish to use. The connect information previously provided is used to retrieve the PeopleSoft Component Interface properties. The Component Interface structure is displayed graphically.
   2.2. Manipulate Template: Here input cells are selected for inclusion on the Data Input and Staging & Submisson sheets. The purpose of each button provided to allow manipulation of the template is more fully described by mousing over the button on the toolbar.
   2.3. New Data Input: This button copies the selected input cells into the Data Input sheet. The Data Input sheet becomes active, and you will be prompted before the structures and data on that sheet are deleted.

3) **Data Input sheet:** Here you enter data values for submission to the PeopleSoft database.
   3.1. Stage Data for Submission: The data entered on the Data Input sheet is then staged in hierarchical form in preparation for submission to the database.

4) **Staging & Submission sheet:** The last step is to format the data and submit to the database.
   4.1. Submit Data: This submits the data to the database. Your PeopleSoft User ID and Password will be required. Each row submitted is marked with the reply from the database which will either be OK, Warning, or Error.
   4.2. Post Results: This posts the status for each row submitted to the database on the Data Input sheet so that data in a status of error can be corrected and resubmitted.

Setting Up Connection Information

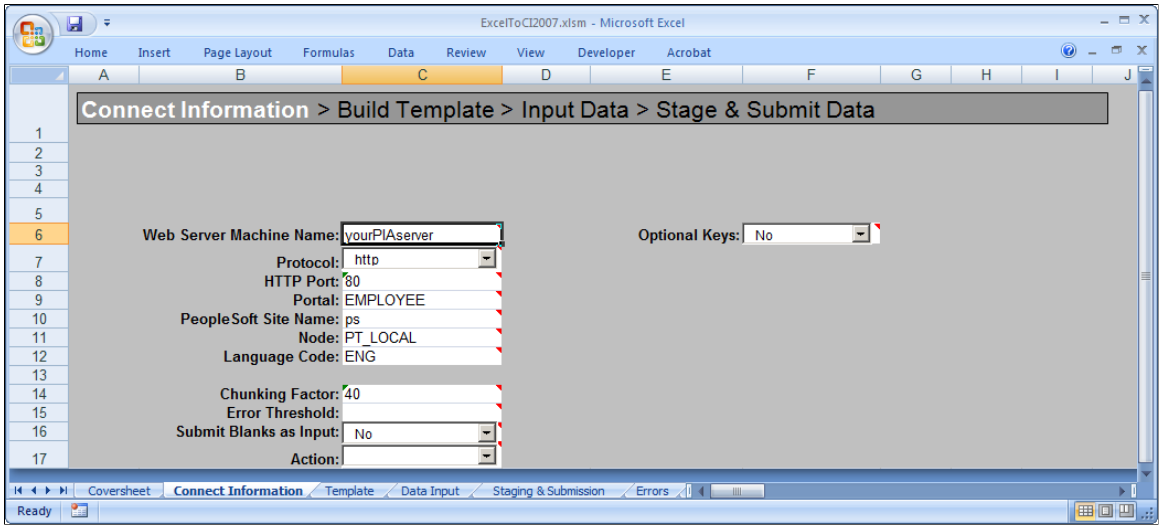
This section discusses how to set up the connection information.

Entering Connection Information

Access the Connect Information tab in ExcelToCI2007.xlsm by clicking the Connect Information tab:

Image: Connect Information tab

This example illustrates the Connect Information tab in the Excel to Component Interface Utility.



The information on this page is required to create a new template or to submit data to the database. You will need to specify environment information as well as information about how data should be transmitted. The Action field will be populated automatically based on your setup and the component interface that the template is associated with.

The initial connection settings will be the PeopleSoft default values. You will need to modify these values for your specific implementation of PeopleSoft. If you are unsure what to enter for these values, check with your system administrator.

The connection options are:

Web Server Machine Name	The name of the PeopleSoft web server to which you are connecting.
Protocol	The protocol used to access the web server. The default is *HTTP*. The preferred protocol is HTTPS.
HTTP Port	The HTTP port number that the web server uses. The default is *80*.
Portal	The name of the portal you are using. *EMPLOYEE* is a default portal shipped with PeopleSoft.
PeopleSoft Site Name	The PeopleSoft site name that you entered when you installed the PeopleSoft Internet Architecture. The default is *ps*.
Node	The PeopleSoft default local node name. The default is *PT_LOCAL*.
Language Code	The code for the language in which the data is submitted to the database. If no language code is specified, the base language is used.

Chunking Factor

The number of rows of data to be transmitted to the database at one time. The default is 40.

Error Threshold

The total number of errors that are permitted before submission to the database ceases. When the error threshold is exceeded, an error message appears and submission to the database stops.

Submit Blanks as Input

When this option is set to *Yes* and a character input field selected for input contains only blank spaces, the field will be included for submission instead of being ignored. This option is set to *No* by default, for backwards compatibility.

If full-width blank space Unicode characters are entered as an input value in ExcelToCI, (this is achieved by using an encoding that supports such Unicode characters) the field will be submitted, the blanks will be sent, and the value will not be trimmed before it is saved to the database.

If regular ASCII blank spaces (also known as half-width characters) are entered as a value for a character field, the field will be submitted, but the value will be trimmed, so an empty string will be saved. In essence, the field value will be cleared.

Action

The value for this field is supplied by the system when the component interface is retrieved from the database. However, you can change the supplied value by selecting it from the Action drop-down list.

The types of actions available are based on the structure of the component interface. The actions are:

- *Create.*

  This option is available if the component interface has create keys. Use this mode when new keys are being added at level 0.

- *Update.*

  This option is available if the component interface does not have create keys. Use this mode if you are adding new children to an existing parent.

- *UpdateData.*

  Use this option to update specific non-key values that already exist. The system uses the keys to locate the row, and when a match is found, the row is updated with new data. If a key match is not found by the system, it displays an error message indicating which collection was missing a key match.

  When using the UpdateData action, you must provide all keys for the collection for the system to modify the data.

Note: If you want to insert an effective-dated collection at Level 1 containing a child collection at Level 2, you may need to use UPDATE to insert the parent row at level 1 and then use UPDATEDATA to insert the child row at level 2. This is because child rows are copied forward from the current effective-dated collection as a result of the insertion of a new effective-dated parent row. These child rows will be updated by the component processor with the new effective date, and may have the same level 2 keys as the Level 2 child row that you are trying to submit from ExcelToCI.

Optional Keys This field is reserved for PeopleSoft internal use

Error Thresholds and Chunking

A running error count is kept for each chunk of data that is being submitted to the database. When the total error count exceeds the error threshold that you specified on the Connection Information tab, submission to the database stops and the system displays an error message. Rows that error out will have a status of *Error* on the data input page and should be corrected. The data submitted to the database before the error threshold was reached will remain in the target database. Rows not yet submitted will be submitted when the data is restaged and submitted.

Translations and Multilingual Support

You can use the Excel to Component Interface utility to upload data from any installed language. The Excel to CI utility delivers separate Excel macros for each delivered language. The macros contain the translated strings used as labels on the main spreadsheet. The macros are located in the appropriate language directory found in the PS_HOME/EXCEL directory.

Enabling Non-English Languages

To enable a non-English language in the Excel to CI Utility:

- Change the language code on the spreadsheet Connect Information tab to the language to the language you want to use.
- Change the reference to the related language macro to be used, as the default macro contains English language strings.

  To change the related language macro, in Excel select the Tools, Macro and right-click on the Visual Basic Editor option. Once in Visual Basic, select Tools, References, and click on the RelLangMcro entry. Change the file to be used to the one with the same name but located inside the translated language directory of your choice. Click OK and then save the change.
- If you are using a language in which a different character set or numeric formatting is used, you need to set the locale of your client machine to match that language. To do so, open Control Panel, Regional settings, and select the correct language and input locale.

Related Links

"PeopleSoft Pure Internet Architecture Fundamentals" (PeopleTools 8.58: Portal Technology)
"Security Basics" (PeopleTools 8.58: Security Administration)

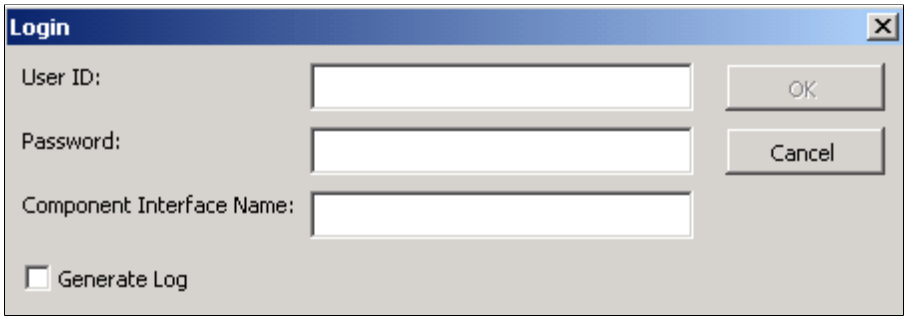
Connecting to the Database to Create a Template and Submit Data

Your PeopleSoft login information is needed for both creating the template and submitting data to the database.

Access the Login dialog box by selecting the Template tab and then clicking the New Template button, or by clicking the Submit Data button on the Staging and Submission tab:

Image: Login dialog box

This example shows the Login dialog box for the Excel-to-Component Interface utility.



The system uses your user ID and password to ensure that you have the correct permissions to access the component interface that you are creating the template on. You must be granted permission to access the component interface that you are using.

User ID/Password	Enter your PeopleSoft user ID and password.
Component Interface Name	Enter the name of the component interface for which you want the template created.
Generate Log	Select the Generate Log check box to create one log file for ExcelToCI2007.xlsm and one for the SOAPTOCI Web Library.

Note: Unless you are troubleshooting errors, you should run the Excel to Component Interface utility without creating log files. Logs should be generated for debugging purposes only.

Related Links

Diagnosing and Resolving Errors

Creating Templates

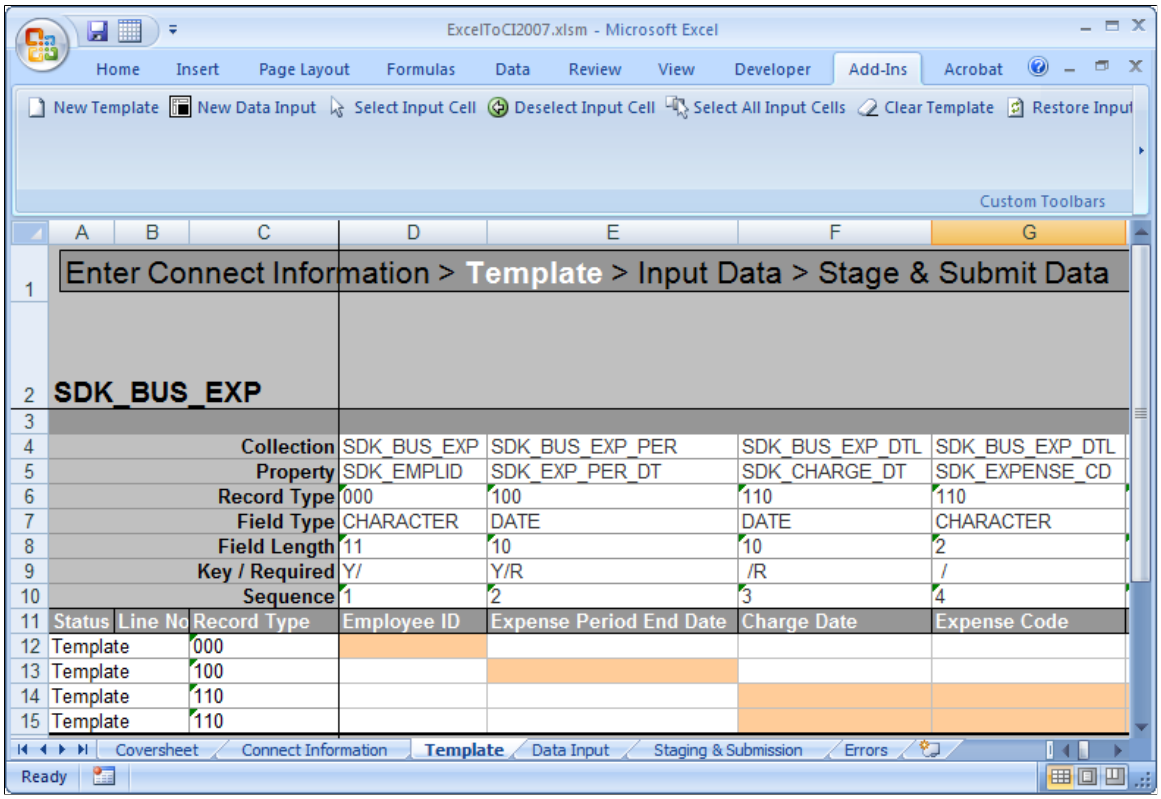
The template page is a graphical representation of the component interface structure that you will be using to load data. The structure of the component interface is retrieved from the database when a new template is built. All of the fields that are exposed through the component interface appear on the template page. Fields that are read-only on the component interface will not appear on the template.

The new template macro builds the parent-child relationship within Microsoft Excel based upon the component interface scroll-level definition. The system adds a new row for each scroll level and assigns a unique identifier to it.

Access the Template tab in ExcelToCI.xlsm to create your template:

Image: Template tab in Excel to Component Interface Utility.

This example illustrates the Template tab in the Excel to Component Interface utility.



Collection The name of the component interface collection. A collection is a property that points to a scroll, rather than a field, in the underlying component for a component interface.

Property The component interface property name. Typically, this is also the name of the field on the page.

Record Type This number represents the parent/child relationship of the records. The level 0 scroll record is always represented by 000. Level 1 scroll records appear with numbers that start with 100 and always have 00 as the last two numbers.

Level 2 scrolls are identified by numbers that start with the identifier of their level 1 parent and end with a 0.

Level 3 scrolls are identified by the first number from the level 1 parent, the second number from its level 2 parent, and then the third number from its own position in the list.

The numbers for each scroll level increments based on the number of records that exist at that level. For example, level 0 would be 000, level 1 would be 100, level 2 would be 110, and so on.

Note: Component interfaces that have more than 10 collections at a given level increments with alphabetic identifiers. For example, 800, 900, A00, and so on.

Field Type The standard PeopleSoft type for the field, for example, *Date, Character,* and so on.

Field Length The length of the field as defined by PeopleSoft. For numeric fields and signed number fields, the length is broken down into integer and decimal positions. For example, a length of 15.3 indicates 15 integer positions and three decimal positions.

Key/Required If the field is a key field, the system will display a *Y* to the left of the forward slash. When the field is not a key, it will be blank. If the field is a required field, the system will display an *R* to the right of the forward slash. When the field is not required, it will be blank. This information comes from the record definition itself.

Note: Fields that are either keys or required *must* be set in order to submit data successfully.

Sequence The sequence number represents the property order in the template.

Status This field displays the load status on the Staging and Submission page.

Line No This corresponds to the line number on the Input Data and the Staging and Submission pages.

Understanding the Template Actions Toolbar

The template actions toolbar is made up of buttons that you use to create and modify a template, as well as create a data input sheet. Each button on the toolbar has help text that describes the purpose and use of each of the buttons when you place the cursor over the button. You can access the toolbar from Add-Ins tab on the Excel standard menu bar in Template, Data Input, and Stage and Submission tabs.

New Template Builds a new template based upon a component interface. The New Template macro builds the parent-child relationship within Microsoft Excel based upon the component interface structure.

When you build a new template, the system prompts you for your sign in information.

New Data Input Builds a new data input sheet based upon the selected input cells. When you build a new data input sheet, the system prompts you as to whether you want to overwrite the existing sheet. If you select *Yes,* a new data input sheet is created, overwriting the former one.

Select Input Cell

Selects an individual cell to be included in the data input sheet. Cells that have been selected as input cells are highlighted in pink.

Select All Input Cells

Selects all properties to be included in the data input as input cells. When a cell is selected as an input cell, it is highlighted in pink.

Restore Input Cells

Restores the template to its original state and clears default values. The fields in the template will be highlighted in gray, indicating that nothing is included for submission.

Insert New Child

Copies the selected row to be inserted as a new child. This creates multiple occurrences of the same record type.

For example, if the selected row has a template identifier of 100, a new row is inserted that also has an identifier of 100 and is an exact duplicate of the selected row.

Note: Use Insert New Child when multiple children must be submitted under the same parent record. Multiple children should not be created at identifier 000.

Include All for Submission

Includes all properties on the spreadsheet to be included for submission to the database. Cells that are included for submission appear only on the Staging and Submission sheet and do not appear on the data input sheet. Properties that are included for submission are highlighted in blue.

Include for Submission

Includes a single property to be included on the Staging and Submission sheet. Properties that use default values from the template must be included for submission. Cells that are included for submission generally are properties that contain default values or properties that you would like to see in the structure of the Staging and Submission sheet. Properties that are included for submission are highlighted in blue.

Deselect Input Cell

Changes a cell that was previously selected as an input cell to a cell that is included for submission. The cell is no longer included on the data input sheet but appears as part of the structure on the Staging and Submission sheet.

Clear Template

Clears all the data and structures on this sheet.

Do Not Include for Submission

Does not include the selected property for submission to the database. If a property is not included for submission, it will not appear in the structure that is submitted to the database on the Staging and Submission sheet. Properties that are not included for submission will appear only on the template worksheet and are not submitted to the database. Properties that are not included for submission are highlighted in gray.

Note: When you create a new template or a new data input sheet, the system clears the existing worksheet of all existing information. If you have a template or data input sheet that you need to save from previous uploads, save a copy of the worksheet before you create a new template or data input sheet.

Entering Data into the Template

When determining which properties to include as input cells and which properties to include for submission, remember that the component interface uses the same business logic and executes the same PeopleCode as if the record were entered online using the page in your PeopleSoft application. To provide the minimal data necessary, these fields must be provided either with default (hard-coded) values or values that you provide using the data input sheet.

Note: You should unit test the template that you created with a few sample entries, and then verify your results before using the interface for mass input. For example, if you forgot to select a property, you will need to build a new data input sheet. If the results of the submission are satisfactory, continue entering data.

Adding a New Child Record

By default, each collection is represented once on the template. To insert copies of a given collection, select that collection and click the Insert New Child button to create a copy of the selected row. The collection that you selected is copied so that you can have two rows under the same parent.

Image: Excel to Component Interface utility

This example illustrates the Template tab in the Excel to Component Interface utility when a child row is added.

	A	B	C	D	E	F	G	H	I
1	Enter Connect Information > **Template** > Input Data > Stage & Submit Data								
2	**SDK_BUS_EXP**								
3									
4			**Collection**	SDK_BUS_EXP	SDK_BUS_EXP_PER	SDK_BUS_EXP_DTL	SDK_BUS_EXP_DTL	SDK_BUS_EXP_DTL	SDK_BUS_EXP_
5			**Property**	SDK_EMPLID	SDK_EXP_PER_DT	SDK_CHARGE_DT	SDK_EXPENSE_CD	SDK_EXPENSE_AMT	SDK_CURRENC
6			**Record Type**	000	100	110	110	110	110
7			**Field Type**	CHARACTER	DATE	DATE	CHARACTER	SIGNED_NUMBER	CHARACTER
8			**Field Length**	11	10	10	2	15.3	3
9			**Key / Required**	Y/R	Y/R	/R	/	/	/R
10			**Sequence**	1	2	3	4	5	6
11	**Status**	**Line No**	**Record Type**	**Employee ID**	**Expense Period End Date**	**Charge Date**	**Expense Code**	**Expense Amount**	**Currency Code**
12	Template		000						
13	Template		100						
14	Template		110						
15	Template		110						

Note: On the data input sheet (when the hierarchy is flattened) you will see duplicate columns where multiple children exist.

Adding Default Values

Some fields have default values associated with them, either in the record definition or at runtime when the record is created on the database. Additionally, many components trigger PeopleCode, which supplies default values, as well. To accept the database default, include the property for submission and the system default will be used.

Some fields may exist for which you want to create your own default. For example, if you want to set the value of a field named Status as of Effective Date to *A* for every row that you submit, enter that value for the field in the template. Then include the cell for submission on the template. The field will not appear on the data input page, but the value will appear in the field on the Submit to Database page. This is useful for effective dates, status fields, set IDs for simple imports, and so on.

When providing values for translate fields or prompt tables, provide the field value rather than the short or long description for the translate value. If you are unsure of the field values, check in the record or field definition in PeopleSoft Application Designer.

Entering Data on the Data Input Sheet

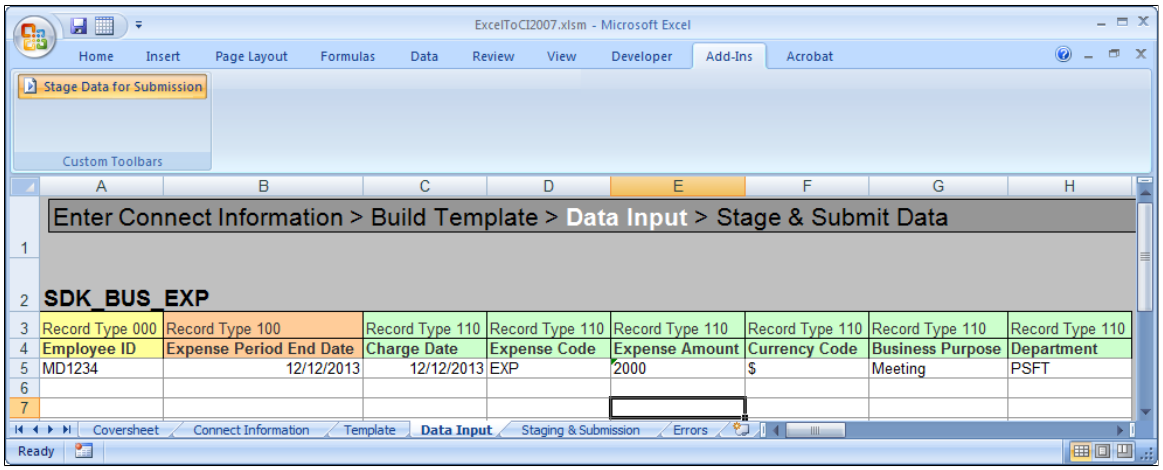
The data input sheet enables you to enter data into the Excel to Component Interface utility so that it can be loaded to the database by using the component interface that you've selected. You can enter data manually or you can cut and paste it from another spreadsheet or third-party application.

Using the Data Input Sheet

Access the data input tab to enter data:

Image: Data Input tab in Excel to Component Interface Utility

This example illustrates the Data Input tab of the Excel to Component Interface utility.



The field labels that appear on the data input sheet are those properties that you selected as input cells on the template. Each scroll level is identified by color. The record type from the template is also displayed for each property.

The system creates default date, datetime, and number formats when it creates the template. You can modify this format by using default cell formatting of Microsoft Excel when entering data, with the exception of the d/m/yy format for dates and datetimes. Instead, always use a d/m/yyyy format for dates and datetimes. To access the formatting feature, select Format > Cells from the Excel menu.

The data input sheet is also used to correct data that failed to submit to the database. Errors that are flagged on the Submit to Database page are posted to the data input page, and when you have corrected them, the items marked in error can be staged again to the Staging and Submission sheet.

Data Input Actions

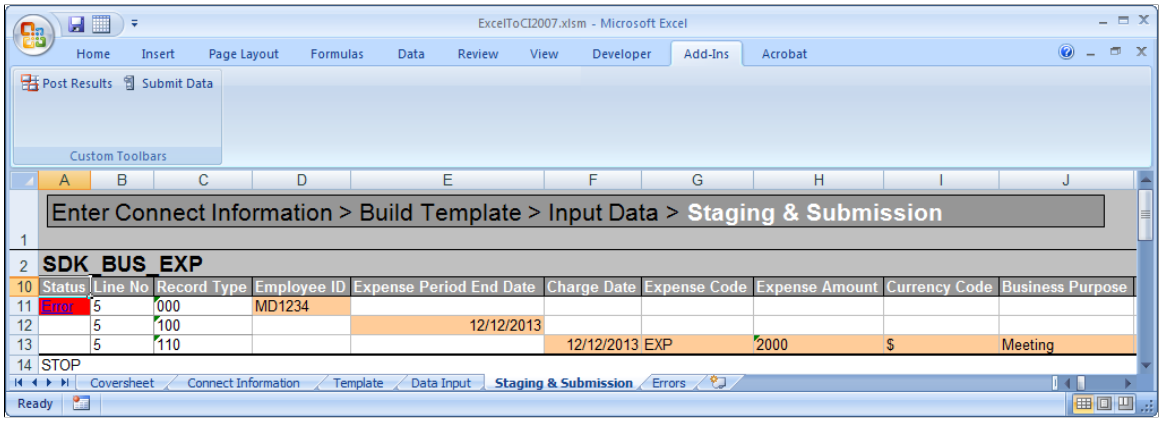
Select the Add-Ins tab in Excel menu bar to access Stage Data for Submission button, which takes the data that you entered on the data input sheet and stages it for submission to the database. When the data is staged, it appears on the Staging and Submission sheet in the hierarchical template structure. At this point, you should check that all fields are populated as expected. When the data is staged, it displays both the data on the data input sheet and the data that you specified as default values.

Viewing the Staged Data

Access the Staging and Submission tab:

Image: Staging & Submission tab in Excel to Component Interface Utility

This example illustrates the Staging and Submission tab of the Excel to Component Interface utility.



Staging and Submission Actions Toolbar

Post Results	The results of the submission are copied to the data input sheet, where you can view the status of each row that is submitted and make any necessary corrections to rows that have the status of *Error*.
Submit Data	The login dialog box appears. You must specify your user ID and password. The system submits the data to the database in the chunks that you specified on the Connection Information sheet. After correcting any errors on the data input sheet, you can submit the data again. The items that had been marked as *Error* will be resubmitted, whereas those marked *OK* and *Warning* will be ignored.

Error When Submitting Existing Keys

If you receive the error message *Row already exists with the specified keys* and you are in CREATE mode, the key already exists at level 0 or is part of the search record.

To verify that the key exists:

1. Open the component interface in PeopleSoft Application Designer.
2. Launch the component interface tester by selecting Tools > Test Component Interface.

   The component interface Tester search dialog box appears. This dialog box displays the keys (in the left-hand columns) for getting, creating, or finding an instance of the component interface.
3. Enter the value for the key that you are testing.
4. Click Get Existing for the key that you are about to add, using the Excel to component interface utility.

   If the Get Keys command returns the key, the key already exists and you must add data by using UPDATE mode.

   If you receive a message that no row exists for the key, then the key does not exist at level 0 and the data should be added by using CREATE mode.

See Testing Component Interfaces.

Correcting and Resubmitting Data

After you submit the data to the database, the result of entering the data into Excel to Component Interface utility appears on the Staging and Submission sheet. If a submission generates an error or a warning, the Status column in the Staging and Submission sheet displays that.

You can read the errors and warnings messages on moving your cursor over the Status column. Also, the utility adds an Errors page which displays consolidated error messages with the respective row numbers.

If you click a cell that displays Error link in the Status column, an HTML page opens. This HTML page also displays the complete list of error and warning messages in a tabular format.

Both the Errors sheet and the HTML page help to display all error messages and warnings on a single sheet. You can correct the input data before staging it for submission to the database again. You can continue this process of correcting errors and resubmitting the data until no errors remain.

Note: Data that was not submitted because the error threshold was reached will have no status. When the data that created the error is corrected on the data input sheet, the data that was not submitted will be staged to the database.

Submission Statuses

Errors received for each record submitted appear in a comment field when you move the cursor over the status column. The records marked *OK* in green have been successfully submitted and cannot be restaged for submission and can be kept as a record of work completed.

One of the following three status values can appear when you submit data to the database:

Ok	The submission to the database finished successfully. The field is highlighted in green. These records cannot be restaged for submission.

Warning The data was saved to the database successfully, but a warning was generated in the process. The field is highlighted in yellow.

Error The data was not saved to the database due to an error. This field is highlighted in red with a hyperlink to a HTML page. The field provides two options to see the error details that the component interface generated.

- Click the Error link to access the HTML page with the error details.
- Place your cursor on the Status field to see the error messages in a comment box.

Note: The same errors are also consolidated in the Errors worksheet in the Excel to Component Interface utility.

Creating SOAP/XML Requests

You can construct a SOAP/XML (Simple Object Access Protocol/Extensible Markup Language) request to create, update, or get component interface rows. The request and response contain component interface data in a SOAP/XML format.

Request Format

The following example shows the request format:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"   xml⇒
ns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <Action__CompIntfc__CIName>
      Tags and Data
    </Action__CompIntfc__CIName>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Valid actions are Create, Get, Update, and Update data.

CIname is the name of the component interface.

Tags and Data contains the tags and data for the component interface row or rows.

Sample Create Request

The following example shows a Create request:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <CREATE__CompIntfc__SUPPORT_DOC_TBL>
    <SUPPORT_DOC_ID>POLICE</SUPPORT_DOC_ID>
    <SUPPORT_DOC>
      <DESCR>Police Report</DESCR>
```



```
      <DESCRSHORT>Police</DESCRSHORT>
    </SUPPORT_DOC>
  </CREATE__CompIntfc__SUPPORT_DOC_TBL>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Get Request

The following example shows a Get request:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <Get__CompIntfc__SDK_BUS_EXP>
    <SDK_EMPLID>8052</SDK_EMPLID>
  </Get__CompIntfc__SDK_BUS_EXP>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Update Request

The following example shows an Update request:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <UPDATE__CompIntfc__SDK_BUS_EXP>
    <SDK_EMPLID>8001</SDK_EMPLID>
    <SDK_BUS_EXP_PER>
      <SDK_EXP_PER_DT>08/14/2002</SDK_EXP_PER_DT>
        <SDK_BUS_EXP_DTL>
          <SDK_CHARGE_DT>08/14/2002</SDK_CHARGE_DT>
          <SDK_EXPENSE_CD>01</SDK_EXPENSE_CD>
          <SDK_EXPENSE_AMT>1234.56</SDK_EXPENSE_AMT>
          <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
          <SDK_BUS_PURPOSE>Client Visit</SDK_BUS_PURPOSE>
          <SDK_DEPTID>10100</SDK_DEPTID>
        </SDK_BUS_EXP_DTL>
    </SDK_BUS_EXP_PER>
  </UPDATE__CompIntfc__SDK_BUS_EXP>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Updatedata Request

The following example shows an Updatedata request:

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Body>
        <UPDATEDATA__CompIntfc__USER_PROFILE>
            <UserID>username</UserID>
            <UserDescription>updated description</UserDescription>
            <EmailAddresses>
                <EmailType>BUS</EmailType>
                <EmailAddress>Updated@updated.com</EmailAddress>
            </EmailAddresses>
        </UPDATEDATA__CompIntfc__USER_PROFILE>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Sending Requests

To send the request, post the XML code to the URL of the PeopleSoft Pure Internet Architecture server with the appropriate path to the iScript on the server.

Note: The PeopleSoft user ID and password must be sent in the SOAP request header with the identifiers of *userid* and *pwd.* You should send the request on a secure site.

Use this format:

```
Protocol(http or https)>://<WebServerMachineName>:<HTTPPort>/psc/ps/<Portal>/<Node>⇒
/s/
WEBLIB_SOAPTOCI.SOAPTOCI.FieldFormula.IScript_SOAPToCI?&disconnect=y&postDataBin=y
```

WebServerMachineName	Machine name of the server.
HTTPPort	Port of the server.
Portal	Portal defined on the PeopleSoft Pure Internet Architecture server.
Node	Node defined on the PeopleSoft Pure Internet Architecture server.

For example,

http://MyWebServer:80/psc/ps/EMPLOYEE/PT_LOCAL/s/
WEBLIB_SOAPTOCI.SOAPTOCI.FieldFormula.IScript_SOAPToCI?disconnect=y&postDataBin=y

Receiving Responses

This section provides examples of response types.

Viewing a Response if a Row Already Exists

This is one example of the error response. The messages vary depending on the type of error.

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <USER_PROFILE>
      <Error-Warning>
        <Message>
          <Type>Error</Type>
          <MessageSetNumber>91</MessageSetNumber>
          <MessageNumber>49</MessageNumber>
          <MessageText>Row already exists with the specified keys.
             {USER_PROFILE} (91,49)</MessageText>
          <ExplainText>A rows already exists in the database with the            s⇒
pecifiedkeys.
             </ExplainText>
        </Message>
      </Error-Warning>
      <Key_information>
        <UserID>PTDMO10</UserID>
      </Key_information>
```



```
    </USER_PROFILE>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Viewing a Sample Get Request and Response

The following XML code gets an SDK_BUS_EXP component interface row for an employee with an employee ID of 8052:

```
<?xml version="1.0"?>
<SDK_BUS_EXP action="GET">
         <SDK_EMPLID key="Y">8052</SDK_EMPLID>
      </SDK_BUS_EXP>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The XML response for this employee is:

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
    <SDK_BUS_EXP>
      <SDK_BUS_EXP_PER>
        <SDK_EMPLID>8052</SDK_EMPLID>
        <SDK_EXP_PER_DT>2000-11-09</SDK_EXP_PER_DT>
        <SDK_BUS_EXP_DTL>
          <SDK_EMPLID>8052</SDK_EMPLID>
          <SDK_EXP_PER_DT>2000-11-09</SDK_EXP_PER_DT>
          <SDK_CHARGE_DT />
          <SDK_EXPENSE_CD />
          <SDK_EXPENSE_AMT>0</SDK_EXPENSE_AMT>
          <SDK_CURRENCY_CD>USD</SDK_CURRENCY_CD>
          <SDK_BUS_PURPOSE />
          <SDK_DEPTID />
        </SDK_BUS_EXP_DTL>
      </SDK_BUS_EXP_PER>
    </SDK_BUS_EXP>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Diagnosing and Resolving Errors

This section discusses how to diagnose and resolve errors in client environments.

Viewing Log Files

If you select the check box to create log files when building a template or submitting to the database, two log files are created—one that logs the activity of ExcelToCI2007.xlsm and the other that logs the SOAPTOCI Web Library.

The log for ExcelToCI2007.xlsm is created in the temp directory on the workstation running the Microsoft Excel spreadsheet.

The log for the Web Library, *SOAPTOCI<unique_number>.log*, is created on the application server in the <PS_CFG_HOME> directory. This file contains both the SOAP request and the SOAP response.

Log files are written for each chunk of data that is sent to the database.

Viewing HTML Page

The HTML page displays a complete list of errors and warnings generated in Staging & Submission sheet of the Excel to Component Interface utility. The errors and warnings are displayed in a tabular format. You can click any Error link in the Status column of Staging & Submission sheet to access this page.

See Correcting and Resubmitting Data

Resolving Error Messages for Client Environments

The following table lists common errors and error messages and their possible resolutions.

Error Message	***Possible Resolution***
Component not correctly registered	Reinstall Visual Basic 6.0 SP5: Run-Time Redistribution Pack found on the Microsoft download site.
ActiveX component not correctly registered (Error 336)	Reinstall Visual Basic 6.0 SP5: Run-Time Redistribution Pack found at the Microsoft downloads website.
Error Number: -2147024770 Description: Automation error. The specified module could not be found.	Perform the following steps: 1. Open Windows Explorer. 2. Navigate to c:\winnt\system32 directory and locate msxml6.dll. 3. Right-click the DLL and select Register COM Server. The message *DLLRegisterServer in c:\winnt \system32\msxml6.dll succeeded.* will appear. 4. Click OK.
Error Number: 429 Description: ActiveX component can't create object.	Perform the following steps: 1. Open Windows Explorer. 2. Navigate to c:\winnt\system32 directory and locate msxml4.dll. 3. Right-click the DLL and select Register COM Server. The message *DLLRegisterServer in c:\winnt \system32\msxml6.dll succeeded.* will appear. 4. Click OK.

Error Message	*Possible Resolution*
Error Number -214722099 Description: Automation error in the dll.	Perform the following steps: 1. Location and open the file ExcelToCI2007.xlsm. 2. Press Alt + F11 to open the Microsoft Visual Basic Editor. 3. Select Tools > Add references. 4. Deselect anything that begins with Microsoft XML. 5. Browse for c:\winnt\system32msxml6.dll and click OK. 6. Select that version of msxml and click OK. 7. Click Save.
Not Authorized (90,6)	The user who is trying to access the component interface from ExcelToCI does not have access to the component interface. Please provide access using PeopleTools Security.

Appendix A

Creating Component Interface-Based Services

Understanding Generating Component Interfaced-Based Services

PeopleSoft Integration Broker enables you to take an existing component interface and create a service that can invoke the component interface.

Further, it creates service operations, including request messages and response messages (if appropriate). The system creates an inbound any-to-local routing for the service operation version, as well as handlers for each method you choose to include in the service.

All service operations you generate from component interfaces are synchronous service operations.

After you create the service operation you can access the service definition to view and capture the WSDL.

Related Links

"Understanding Creating Component Interface-Based Services" (PeopleTools 8.58: Integration Broker)

Appendix B

Using Services to Validate Prompt Table and Translate Field Values

Understanding Validating Prompt Table and Translate Field Values

PeopleSoft delivers a a PTLOOKUP service and a REST-based PTLOOKUP_REST service that enables integration partners to retrieve lists of valid/allowable values for prompt and translate (XLAT) fields from PeopleSoft components on which component interfaces are based, allowing them to validate their client application data against PeopleSoft data.

PTLOOKUP Service

To use the PTLOOKUP service, you generate a WSDL document for this service using the Provide Web Service wizard and furnish your integration partner the WSDL document. The third-party integration partner uses the provided request message shape contained in the WSDL document to specify the field values to validate. They then send the request message to the PeopleSoft system to invoke the service. The PeopleSoft system returns a response message to the integration partner with the field values requested.

The PTLOOKUP service contains two service operations:

PTLOOKUPPROMT.v1	Use this service operation to return prompt table field values for prompt tables contained in a component.
PTLOOKUPXLAT.v1	Use this service operation to return translate (XLAT) field values for translate fields contained in a component.

Each service operation is synchronous and is delivered with a request message, a response message, a handler, and a routing. The delivered metadata for these service operations is described elsewhere in this section.

The service operations take as their primary inputs the value being validated, the name of the table, and the field name against which to compare. The service operations compare the input value against the lookup table and return the result of the validation test.

PTLOOKUP_REST Service

To use the PTLOOKUP_REST service, you generate a WADL document for this service using the Provide Web Service wizard and furnish the WADL document to your integration partner to generate The third-party integration partner uses the provided request document type message shape to specify the field values to validate. They then use the request URL contained in the WADL document to invoke the service on the PeopleSoft system. The PeopleSoft system returns a response message to the integration partner with the field values requested.

The PTLOOKUP_REST service contains two service operations:

PTLOOKUPPROMPT_REST_GET.v1 Use this service operation to return prompt table field values for prompt tables contained in a component.

PTLOOKUPXLAT_REST_GET.v1 Use this service operation to return translate (XLAT) field values for translate fields contained in a component.

Each service operation is synchronous and is delivered with pre-defined URI templates, a document template, request and response messages, and any-to-local routing definitions. The delivered metadata for these service operations is described elsewhere in this section.

The service operations take as their primary inputs the value being validated, the name of the table, and the field name against which to compare. The service operations compare the input value against the lookup table and return the result of the validation test.

Prerequisites for Validating Prompt Table and Translate Field Values

The following list outlines prerequisites for using the PTLOOKUP service to validate prompt table and translate table field values:

- The PeopleSoft system must have Integration Broker configured and running.
- Integration partners must know and provide the field names and table names for which they are retrieving validation information.
- Integration partners must have access to:
    - PeopleSoft Application Designer to inspect PeopleSoft component, record, and field information and properties.
    - PeopleSoft Integration Broker or another services-oriented architecture environment configured and running

Validating Prompt Table Field Values

This section discusses how to use the PTLOOKUPPROMPT service operation and the PTLOOKUPPROMPT_REST_GET service operation to validate prompt table field values.

Understanding Validating Set Control Fields

Use service operation security permission lists to secure access to these service operations.

Requests must be sent to the PeopleSoft system using SSL or TLS.

Moreover, because prompt table field values can contain sensitive or confidential information, such as salary grades or order categories. Access to the prompt tables targeted by the *PTLOOKUPPROMPT* service operation and the *PTLOOKUPPROMPT_REST_GET* service operation are restricted by the requirement that they be added to a query access tree for which the user issuing the service request must be granted permission.

Using the PTLOOKUPPROMPT Service Operation

The *PTLOOKUPPROMPT* service operation is a non-REST-based service operation that you can use to validate prompt table field values. To access the service operation, select PeopleTools >Integration Broker >Integration Setup >Service Operations and select the *PTLOOKUPPROMPT* service operation.

PTLOOKUPPROMPT is a restricted service operation that is delivered with the following metadata:

Metadata Type	***Name***	***Description***
Request Message	PTLOOKUPPROMPT.V1	PTLOOKUPPROMPT.V1 is a document-based message.
Response Message	PTLOOKUPRESP.V1	PTLOOKUPRESP.V1 is a document-based message.
Handler	REQUESTHNDLR	REQUESTHNDLR is an OnRequest handler that is implemented using an application class. The application class package delivered is PT_IB_LOOKUP and the class ID is RequestHandler.
Routing	System generated.	PeopleSoft delivers a system-generated synchronous any-to-local routing for this service operation.
Listening Connector	PeopleSoftServices	The default listening connector.

Using the PTLOOKUPPROMPT_REST_GET Service Operation

The *PTLOOKUPPROMPT_REST_GET* is a REST-based service operation that you can use to validate prompt table field values. To access the service operation, select PeopleTools >Integration Broker >Integration Setup >Services. Select the *PTLOOKUP_REST* service. The Services page appears. In the Existing Operations section click *PTLOOKUPPROMPT_REST_GET.*

The *PTLOOKUPPROMPT_REST_GET* service operation is delivered with the following metadata:

Metadata Type	***Name***	***Description***
URI Templates	NA	The service operation is delivered with six (6) pre-defined URI templates. The templates are listed after this table.
Document Template	PTLOOKUPPROMPTTMPL.V1	The Document type message. The document message has elements defined with names used for value replacement within the URI template(s).
Request Message	NA	In a REST-based service operation a request is made by sending a URL based on one of the URI templates.

Metadata Type	*Name*	*Description*
Response Message	PTLOOKUPRESP.V1	*PTLOOKUPRESP.V1* is a document-based message.
Fault Message	PT_LOOKUP_RESTFAULT.v1	*PT_LOOKUP_RESTFAULT.v1* is a document-based message.
Handler	REQUESTHNDLR	*REQUESTHNDLR* is an OnRequest handler that is implemented using an application class. The application class package delivered is *PT_IB_LOOKUP* and the class ID is *RequestHandler.*
Routing	System generated.	PeopleSoft delivers a system-generated synchronous any-to-local routing for this service operation.
Listening Connector	RESTListeningConnector	The default listening connector.

The following URI templates are delivered with this service operation:

- `Lookup/{LookupRecName}/{LookupFieldName}?fieldvalue={LookupFieldValue}`
- `Lookup/{LookupRecName}/{SetIDValue}/{LookupFieldName}?fieldvalue={LookupFieldValue}`
- `Lookup/{LookupRecName}/{LanguageCode}/{SetIDValue}/{LookupFieldName}?fieldvalue={LookupFieldValue}`
- `Lookup/{LookupRecName}/{EffectiveDate}/{LanguageCode}/{SetIDValue}/{LookupFieldName}?fieldvalue={LookupFieldValue}`
- `Lookup/{LookupRecName}/{EffectiveDate}/{LanguageCode}/{SetIDValue}/{DescrFieldName }/{LookupFieldName}?fieldvalue={LookupFieldValue}`
- `Lookup/{LookupRecName}/{EffectiveDate}/{LanguageCode}/{SetIDValue}/{SetControlFieldValue}/{DescrFieldName }/{LookupFieldName}?fieldvalue={LookupFieldValue}`

Validating Translate (XLAT) Field Values

This section discusses how to use the *PTLOOKUPXLAT* service operation and the *PTLOOKUPXLAT_REST_GET* service operation to validate XLAT field values.

Note: XLAT values are effective-dated, and only the values marked as *Active* are used for validation.

Understanding Translate (XLAT) Table Entries

XLAT table entries associated with a field definition include the following attributes:

Attribute	*Description*
FIELDNAME	Field name, such as ABSENCE_TYPE.
LANGUAGE_CD	Language code.
FIELDVALUE	Value for the field; it must be between 1 and 4 characters long.
EFFDT	Effective date.
VERSION	Internal version number (system-maintained).
EFF_STATUS	The status of the field, active or inactive.
XLATLONGNAME	Thirty-character description; used as a label on pages and reports.
XLATSHORTNAME	Ten-character description; used as a label on pages and reports.

Understanding Security When Validating Translate (XLAT) Field Values

Use service operation security permission lists to secure access to this service operation.

Requests must be sent to the PeopleSoft system using SSL or TLS.

Using the PTLOOKUPXLAT Service Operation

The PTLOOKUPXLAT service operation is a non-REST service operation that you can use to validate prompt table field values. To access the service operation, select PeopleTools >Integration Broker >Integration Setup >Service Operations and select the *PTLOOKUPXLAT* service operation.

PTLOOKUPXLAT is a restricted service operation that is delivered with the following metadata:

Metadata Type	*Name*	*Comments*
Request Message	PTLOOKUPXLAT.V1	PTLOOKUPXLAT.V1 is a document-based message.
Response Message	PTLOOKUPXLATRESP.V1	PTLOOKUPXLATRESP.V1 is a document-based message.
Handler	REQUESTHNDLR	REQUESTHNDLR is an OnRequest handler that is implemented using an application class. The application class package delivered is PT_IB_LOOKUP and the class ID is RequestHandler.

Metadata Type	*Name*	*Comments*
Routing	System generated.	PeopleSoft delivers a system-generated synchronous any-to-local routing for this service operation.
Listening Connector	PeopleSoftServices	The default listening connector.

Using the PTLOOKUPXLAT_REST_GET Service Operation

The *PTLOOKUPXLAT_REST_GET* service operation is a REST-based service operation that you can use to validate prompt table field values. To access the service operation, select PeopleTools >Integration Broker >Integration Setup >Services. The Service page appears. In the Existing Operation section click the *PTLOOKUPXLAT_REST_GET* service operation.

The *PTLOOKUPXLAT_REST_GET* service operation is delivered with the following metadata:

Metadata Type	*Name*	*Description*
URI Templates	NA	The service operation is delivered with four (4) pre-defined URI templates. The templates are listed after this table.
Document Template	PTLOOKUPXLAT.V1	The Document type message. The document message has elements defined with names used for value replacement within the URI template(s).
Request Message	NA	In a REST-based service operation a request is made by sending a URL based on one of the URI templates.
Response Message	PTLOOKUPRESP.V1	*PTLOOKUPRESP.V1* is a document-based message.
Fault Message	PT_LOOKUP_RESTFAULT.v1	*PT_LOOKUP_RESTFAULT.v1* is a document-based message.
Handler	REQUESTHNDLR	*REQUESTHNDLR* is an OnRequest handler that is implemented using an application class. The application class package delivered is *PT_IB_LOOKUP* and the class ID is *RequestHandler.*
Routing	System generated.	PeopleSoft delivers a system-generated synchronous any-to-local routing for this service operation.
Listening Connector	RESTListeningConnector	The default listening connector.

The following URI templates are delivered with this service operation:

- `XLAT_Lookup/{LookupFieldName}?fieldVal={LookupFieldValue}`

- `XLAT_Lookup/{LookupFieldName}/{DescrFieldName}/{EffectiveDate}`
- `XLAT_Lookup/{LookupFieldName}/{LanguageCode}/{DescrFieldName}/{EffectiveDate}`
- `XLAT_Lookup/{LookupFieldName}/{LanguageCode}/{EffectiveDate}/{DescrFieldName}?fieldVal={LookupFieldValue}`

Using Messages to Request Valid Prompt Field and Translate (XLAT) Field Values

The request message for either service operation is a document type message, and includes one or more sets of the following as inputs associated with the look-up operation:

Element	***Description***
LookupRecName	Populate this element with the prompt table that contains the data to validate. This element appears only when performing a prompt table lookup and working with the PTLOOKUPPROMPT message. When you perform an XLAT lookup, there is only one XLAT table for the entire PeopleSoft database and the table name, PSXLATITEM, is fixed. As a result you do not need to provide the table name when performing an XLAT lookup.
LookupFieldName	Populate this element with the field name to validate.
LookupFieldValue	(Optional) Populate this element with the name of the descriptor field. The descriptor field describes the purpose of the record, not the field. The valid values are: • A specific value to look up. • Blank (empty). This will return a list of all possible values. • A wildcard (%) value. For example, entering *A%* will return all results that start with the letter A.
DescrFieldName	(Optional) Populate this element with the field name description.
LanguageCode	(Optional) Populate this element with the language code.
EffectiveDate	(Optional) Populate this element with the effective date.

Element	Description
SetControlFieldValue	(Optional) Appears only in the PTLOOKUPPROMPT.V1 request message for validating prompt table field values. If you are validating a prompt table field value that is controlled by a set control field, enter the set control field value in the SetControlFieldValue element in the request message. The system uses this information to extract the name of the SETID field used to partition the data in the table. If you do not enter the set control field value, you must enter the set ID value in the SetIDValue element in the request message. The SetIDValue element is described elsewhere in this table. If you enter a set control field value and a set ID value, the system uses the set control field value to locate the field name.
SetIDValue	(Optional) Appears only in the PTLOOKUPPROMPT.V1 request message for validating prompt table field values. If you are validating a prompt table field value that is controlled by a set control field, enter the set ID value in the SetIDValue element. If you also enter a set control field value in the request message, as described elsewhere in this table, the system uses that value, not the set ID value, to locate the field name.
FldName	(Optional) Appears only in the PTLOOKUPPROMPT.V1 request message for validating prompt table field values. If you are validating a prompt table field use this element in conjunction with the FldNameValue element to return name/value pairs for the field.
FldNameValue	(Optional) Appears only in the PTLOOKUPPROMPT.V1 request message for validating prompt table field values. If you are validating a prompt table field use this element in conjunction with the FldName element to return name/value pairs for the field.

Examples of response messages are provided elsewhere in this appendix.

Using Response Messages to Retrieve Valid Prompt Field and Translate (XLAT) Field Values

The following table lists the elements contained in the response message and the information contained in each:

Element	Description
RespVal	This element is populated with the response value.

Element	*Description*
RespDescr	This element is populated with a description of the response value.

The response value that the PeopleSoft system sends back is based on one of the following possible conditions:

Condition	*Response*
Input value is a perfect match.	The value that was matched.
Input value is a partial match.	List of matched values.
Input value is blank (empty).	List of all values.
Input value not matched.	List of all values.
Prompt table name or prompt field are incorrect.	Error message.

Examples of response messages are provided elsewhere in this appendix.

Examples: Validating Prompt Field and Translate (XLAT) Field Values

This section provides examples of request and response messages for the validating prompt field and translate (XLAT) field values using the PTLOOKUPPROMPT and PTLOOKUPXLAT service operations.

Note: The examples in this section are for non-REST requests and responses only.

Example 1: Validating a Translate (XLAT) Field

The following code example shows a request message sent to a PeopleSoft system as part of the PTLOOKUPXLAT service operation to obtain a list of valid field values and their descriptions for the CERTTYPE field. Note that the LookupFieldValue element is empty (< >). As a result, the PeopleSoft system will return a list of all valid values for the field:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:s⇒
oapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://schemas.xmlsoa⇒
p.org/ws/2003/03/addressing/" xmlns:xsd="http://www.w3.org/2001/XMLSchema/" xmlns:x⇒
si="http://www.w3.org/2001/XMLSchema-instance/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soap:mustUnderstand="1" xmlns:soap="http://schemas.     xmlsoap.⇒
org/wsdl/soap/" xmlns:wsse="http://docs.oasis-open.org/wss/     2004/01/oasis-20040⇒
1-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>username</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
```



```
  <soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <XLAT_Lookup xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/     PT_Lo⇒
okup.XLAT_Lookup.V1">
      <LookupFieldName>CERTTYPE</LookupFieldName>
      <LookupFieldValue></LookupFieldValue>
      <DescrFieldName>XLATSHORTNAME</DescrFieldName>
      <LanguageCode></LanguageCode>
      <EffectiveDate>2010-01-03</EffectiveDate>
    </XLAT_Lookup>
  </soapenv:Body>
</soapenv:Envelope>
```

The following code example shows the response message that the PeopleSoft system returns to the integration partner. The returned field values are returned in the <RespVal> and <RespDescr> elements, as highlighted in the example:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <LookupResponse xmlns="http://xmlns.oracle.com/Enterprise/
        Tools/schemas/PT_Lookup.LookupResponse.V1">
            <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/
            Tools/schemas/PT_Lookup.ResponseComp.V1">
                <RespVal>USER</RespVal>
                <RespDescr>User</RespDescr>
            </ResponseComp>
            <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/
            Tools/schemas/PT_Lookup.ResponseComp.V1"><RespVal>NODE</RespVal>
                <RespDescr>Node</RespDescr>
            </ResponseComp>
            <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/
            Tools/schemas/PT_Lookup.ResponseComp.V1"> <RespVal>CERT</RespVal>
                <RespDescr>Cert</RespDescr>
            </ResponseComp>
            <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/
            Tools/schemas/PT_Lookup.ResponseComp.V1"><RespVal>ROOT</RespVal>
                <RespDescr>Root CA</RespDescr>
            </ResponseComp>
        </LookupResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Example 2: Performing a Prompt Table Lookup with a Field Value Wildcard

The following code example shows a request message sent to a PeopleSoft system as part of the PTLOOKUPPROMPT service operation to perform a prompt table lookup on the Country table using a wildcard (%) on the field value to find country names that begin with the letter U:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http:
//schemas.xmlsoap.org/ws/2003/03/addressing/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soap:mustUnderstand="1" xmlns:soap="http://schemas.xmlsoap.
    org/wsdl/soap/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
    200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>username</wsse:Username>
        <wsse:Password>password/wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
```



```
    <Lookup xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
    PT_Lookup.Prompt_Lookup.V1">
      <LookupRecName>COUNTRY_TBL</LookupRecName>
      <LookupFieldName>COUNTRY</LookupFieldName>
      <LookupFieldValue>U%</LookupFieldValue>
      <DescrFieldName>DESCR</DescrFieldName>
      <LanguageCode></LanguageCode>
      <EffectiveDate></EffectiveDate>
    </Lookup>
  </soapenv:Body>
</soapenv:Envelope>
```

The following code example shows the response message that the PeopleSoft system returns to the integration partner. The returned field values are returned in the <*RespVal*> and <*RespDescr*> elements, as highlighted in the example:

```
<?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv=
"http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://schemas.
xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body><LookupResponse xmlns="http://xmlns.oracle.com/Enterprise/
   Tools/schemas/PT_Lookup.LookupResponse.V1">
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1">
         <RespVal>UMI</RespVal>
         <RespDescr>US Minor Outlying Islands</RespDescr>
      </ResponseComp>
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1"><RespVal>UGA</RespVal>
         <RespDescr>Uganda</RespDescr>
      </ResponseComp>
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1"><RespVal>UKR</RespVal>
         <RespDescr>Ukraine</RespDescr>
      </ResponseComp>
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1"><RespVal>USA</RespVal>
         <RespDescr>United States</RespDescr>
      </ResponseComp>
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1"><RespVal>URY</RespVal>
         <RespDescr>Uruguay</RespDescr>
      </ResponseComp>
      <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.ResponseComp.V1"><RespVal>UZB</RespVal>
         <RespDescr>Uzbekistan</RespDescr>
      </ResponseComp>
    </LookupResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 3: Filtering Field Values by Name/Value Pairs

The following code example shows a request message sent to a PeopleSoft system from an integration partner as part of the PTLOOKUPPROMPT service operation to obtain a list of field values from the Currency table of currencies from Argentina that start with A by using name/value pair as additional filter:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"xmlns:wsa="http:
//schemas.xmlsoap.org/ws/2003/03/addressing/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soap:mustUnderstand="1" xmlns:soap="http://schemas.
    xmlsoap.org/wsdl/soap/" xmlns:wsse="http://docs.oasis-open.org/wss/
```



```
    2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>username/wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <Lookup xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
   PT_Lookup.Prompt_Lookup_All.V1">
     <LookupRecName>CURRENCY_CD_TBL</LookupRecName>
     <LookupFieldName>CURRENCY_CD</LookupFieldName>
     <LookupFieldValue>A%</LookupFieldValue>
     <DescrFieldName>DESCR</DescrFieldName>
     <LanguageCode></LanguageCode>
     <EffectiveDate/>
     <SetControlFieldValue/>
     <SetIDValue/>
     <NameValPair xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
     PT_Lookup.NameValPair.V1"><FldName>COUNTRY</FldName>
        <FldVal>ARG</FldVal>
     </NameValPair>
  </Lookup>
  </soapenv:Body>
</soapenv:Envelope>
```

The following code example shows the response message that the PeopleSoft system returns to the integration partner. The returned field values are returned in the <RespVal> and <RespDescr> elements, as highlighted in the example:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
" xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Body>
        <LookupResponse xmlns="http://xmlns.oracle.com/Enterprise/Tools/
        schemas/PT_Lookup.LookupResponse.V1">
            <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/
            schemas/PT_Lookup.ResponseComp.V1">
                <RespVal>ARS</RespVal>
                <RespDescr>Argentine Peso</RespDescr>
            </ResponseComp>
        </LookupResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Example 4: Specifying Set Control Field Values to Validate Field Values Controlled by Set Control Fields

The following code example shows a request message sent to a PeopleSoft system from an integration partner as part of the *PTLOOKUPPROMPT* service operation to obtain a list of valid field values for the *VENDOR_ID* prompt field, a field controlled by a set control field.

When you provide the set control field value, PeopleSoft uses Set ID indirection (via the GetSetID built-in function) to obtain the set ID value, and uses it to filter results during lookup.

This example shows specifying the set control field value to obtain the values for the field:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
"xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http:
//schemas.xmlsoap.org/ws/2003/03/addressing/" xmlns:xsd="http://www.w3.org/
2001/XMLSchema/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/">
```



```
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soap:mustUnderstand="1" xmlns:soap="http://schemas.xmlsoap.
    org/wsdl/soap/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
    oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>username</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <Lookup xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
  PT_Lookup.Prompt_Lookup_All.V1">
    <LookupRecName>VENDOR</LookupRecName>
    <LookupFieldName>VENDOR_ID</LookupFieldName>
    <LookupFieldValue>TPDENTIST</LookupFieldValue>
    <DescrFieldName>VENDOR_NAME_SHORT</DescrFieldName>
    <LanguageCode/>
    <EffectiveDate/><SetControlFieldValue>US001</SetControlFieldValue>
    <SetIDValue/>
    <NameValPair xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
    PT_Lookup.NameValPair.V1">
        <FldName/>
        <FldVal/>
      </NameValPair>
    </Lookup>
   </soapenv:Body>
</soapenv:Envelope>
```

The following code example shows the response message that the PeopleSoft system returns to the integration partner. The returned field values are returned in the <*RespVal*> and <*RespDescr*> elements, as highlighted in the example.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http:
//www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
    <soapenv:Body>
      <LookupResponse xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
      PT_Lookup.LookupResponse.V1">
         <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
         PT_Lookup.ResponseComp.V1">
              <RespVal>TPDENTIST</RespVal>
              <RespDescr>SMILEWELL-001</RespDescr>
         </ResponseComp>
      </LookupResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Example 5: Specifying Set Control ID Values to Validate Field Values Controlled by Set ID Values

The following code example shows a request message sent to a PeopleSoft system from an integration partner as part of the *PTLOOKUPPROMPT* service operation to obtain a list of valid field values for the *VENDOR_ID* prompt field, a field controlled by a set control field. This example shows specifying the set control ID value to obtain the field values:

```
<?xml version="1.0"?>
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
"xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http:
//schemas.xmlsoap.org/ws/2003/03/addressing/"xmlns:xsd="http://www.w3.org/
2001/XMLSchema/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance/">
  <soapenv:Header xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <wsse:Security soap:mustUnderstand="1" xmlns:soap="http://schemas.xmlsoap.
     org/wsdl/soap/" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
```



```
     200401-wss-wssecurity-secext-1.0.xsd">
      <wsse:UsernameToken>
        <wsse:Username>username</wsse:Username>
        <wsse:Password>password</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
<soapenv:Body xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <Lookup xmlns="http://xmlns.oracle.com/Enterprise/Tools/schemas/
    PT_Lookup.Prompt_Lookup_All.V1">
        <LookupRecName>VENDOR</LookupRecName>
        <LookupFieldName>VENDOR_ID</LookupFieldName>
        <LookupFieldValue>TPDENTIST</LookupFieldValue>
        <DescrFieldName>VENDOR_NAME_SHORT</DescrFieldName>
        <LanguageCode/>
        <EffectiveDate/>
        <SetControlFieldValue/><SetIDValue>SHARE</SetIDValue>
        <NameValPair xmlns="http://xmlns.oracle.com/Enterprise/Tools/
        schemas/PT_Lookup.NameValPair.V1">
            <FldName/>
            <FldVal/>
         </NameValPair>
     </Lookup>
   </soapenv:Body>
</soapenv:Envelope>
```

The following code example shows the response message that the PeopleSoft system returns to the integration partner. The returned field values are returned in the <*RespVal*> and <*RespDescr*> elements, as highlighted in the example.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
"xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd=
"http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
   <soapenv:Body>
       <LookupResponse xmlns="http://xmlns.oracle.com/Enterprise/Tools/
       schemas/PT_Lookup.LookupResponse.V1">
          <ResponseComp xmlns="http://xmlns.oracle.com/Enterprise/Tools/
          schemas/PT_Lookup.ResponseComp.V1">
              <RespVal>TPDENTIST</RespVal>
              <RespDescr>SMILEWELL-001</RespDescr>
           </ResponseComp>
        </LookupResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

